Interactive Web Applications with RStudio and Shiny

Randall Pruim
Calvin College

December 2015 Big Data + Hadoop MeetUp
Calvin College


HELLO
my name is
Garrett
Randy

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

RStudio

strataconf.com

#StrataHadoop

Interactive Shiny Applications

built on Big Data

Slides at: bit.ly/rday-nyc-strata15

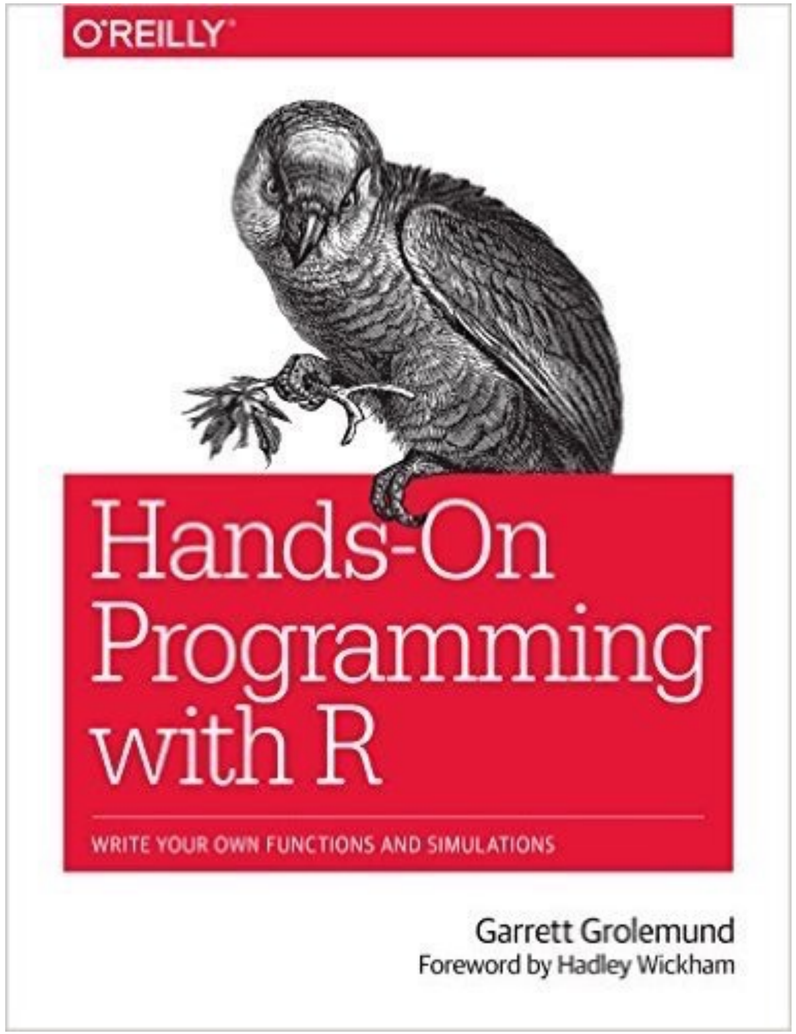
Garrett Grolemund

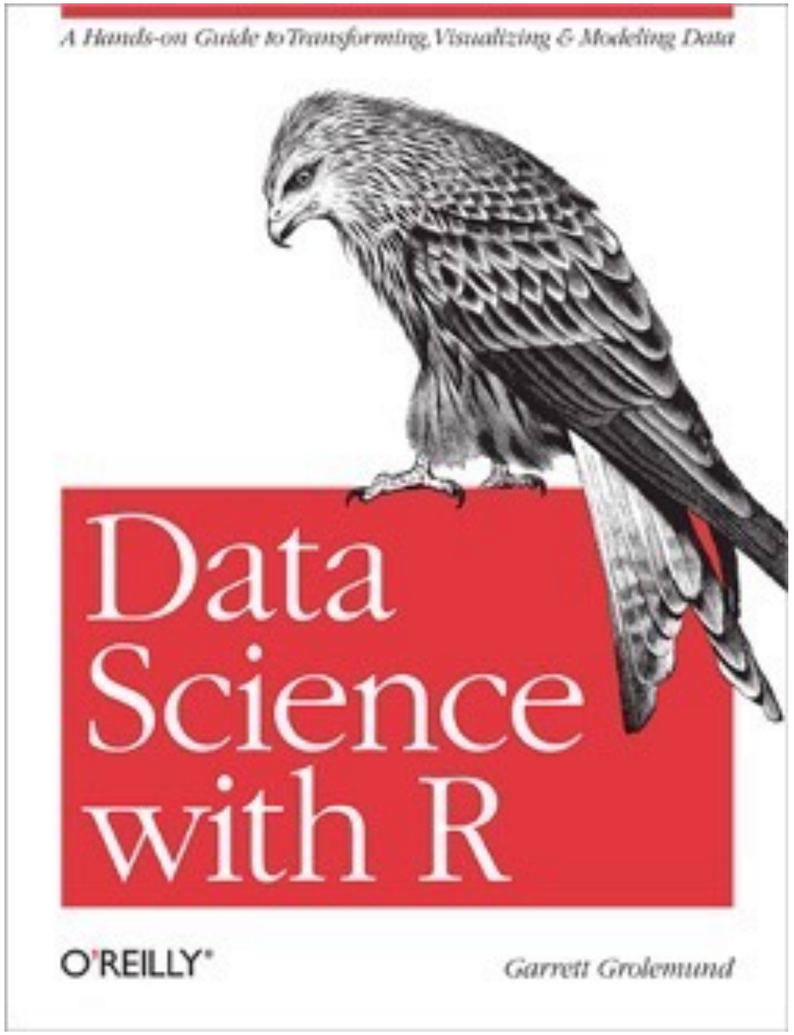
Data Scientist and Master Instructor

September 2015

Email: garrett@rstudio.com

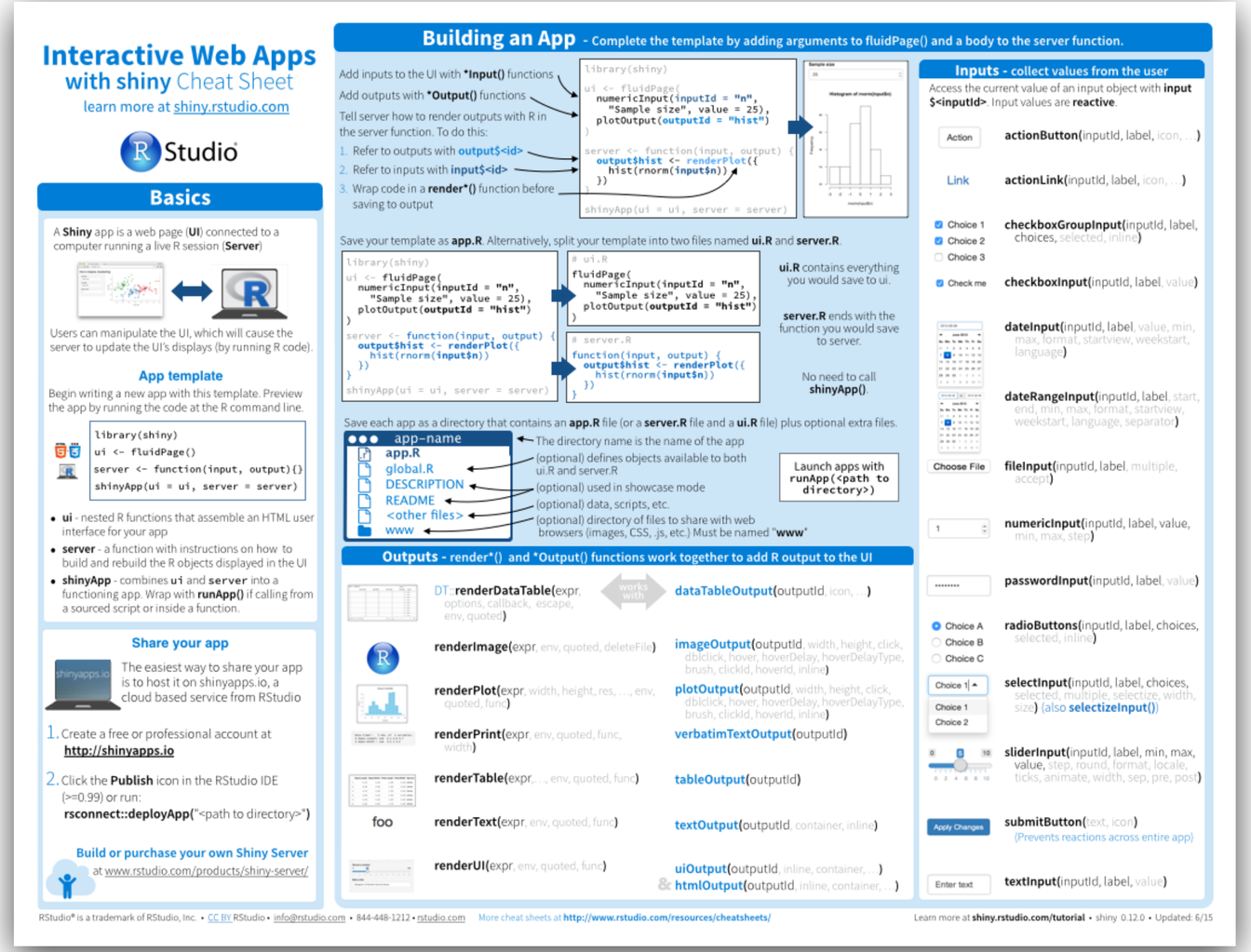
Follow @rstudio


O'REILLY
Hands-On Programming with R
WRITE YOUR OWN FUNCTIONS AND SIMULATIONS
Garrett Grolemund
Foreword by Hadley Wickham


A Hands-on Guide to Transforming, Visualizing & Modeling Data
Data Science with R
O'REILLY
Garrett Grolemund

The Shiny Cheat Sheet

www.rstudio.com/resources/cheatsheets/



Interactive Web Apps with shiny Cheat Sheet

learn more at shiny.rstudio.com

RStudio

Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

App template

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines **ui** and **server** into a functioning app. Wrap with **runApp()** if calling from a sourced script or inside a function.

Share your app

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at **http://shinyapps.io**
2. Click the **Publish** icon in the RStudio IDE (>=0.99) or run: **rsconnect::deployApp("<path to directory>")**

Build or purchase your own Shiny Server at www.rstudio.com/products/shiny-server/

Building an App - Complete the template by adding arguments to fluidPage() and a body to the server function.

Add inputs to the UI with ***Input()** functions

Add outputs with ***Output()** functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with **output$<id>**
2. Refer to inputs with **input$<id>**
3. Wrap code in a **render*()** function before saving to output

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

ui.R contains everything you would save to ui.

```
# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

server.R ends with the function you would save to server.

No need to call **shinyApp()**.

Save each app as a directory that contains an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

- app-name
  - **app.R** — The directory name is the name of the app
  - global.R — (optional) defines objects available to both ui.R and server.R
  - DESCRIPTION — (optional) used in showcase mode
  - README — (optional) data, scripts, etc.
  - <other files> — (optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"
  - www

Launch apps with `runApp(<path to directory>)`

Outputs - render*() and *Output() functions work together to add R output to the UI

	works with	
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)		**dataTableOutput**(outputId, icon, ...)
renderImage(expr, env, quoted, deleteFile)		**imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
renderPlot(expr, width, height, res, ..., env, quoted, func)		**plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
renderPrint(expr, env, quoted, func, width)		**verbatimTextOutput**(outputId)
renderTable(expr, ..., env, quoted, func)		**tableOutput**(outputId)
renderText(expr, env, quoted, func)		**textOutput**(outputId, container, inline)
renderUI(expr, env, quoted, func)		**uiOutput**(outputId, inline, container, ...) & **htmlOutput**(outputId, inline, container, ...)

Inputs - collect values from the user

Access the current value of an input object with **input $<inputId>**. Input values are **reactive**.

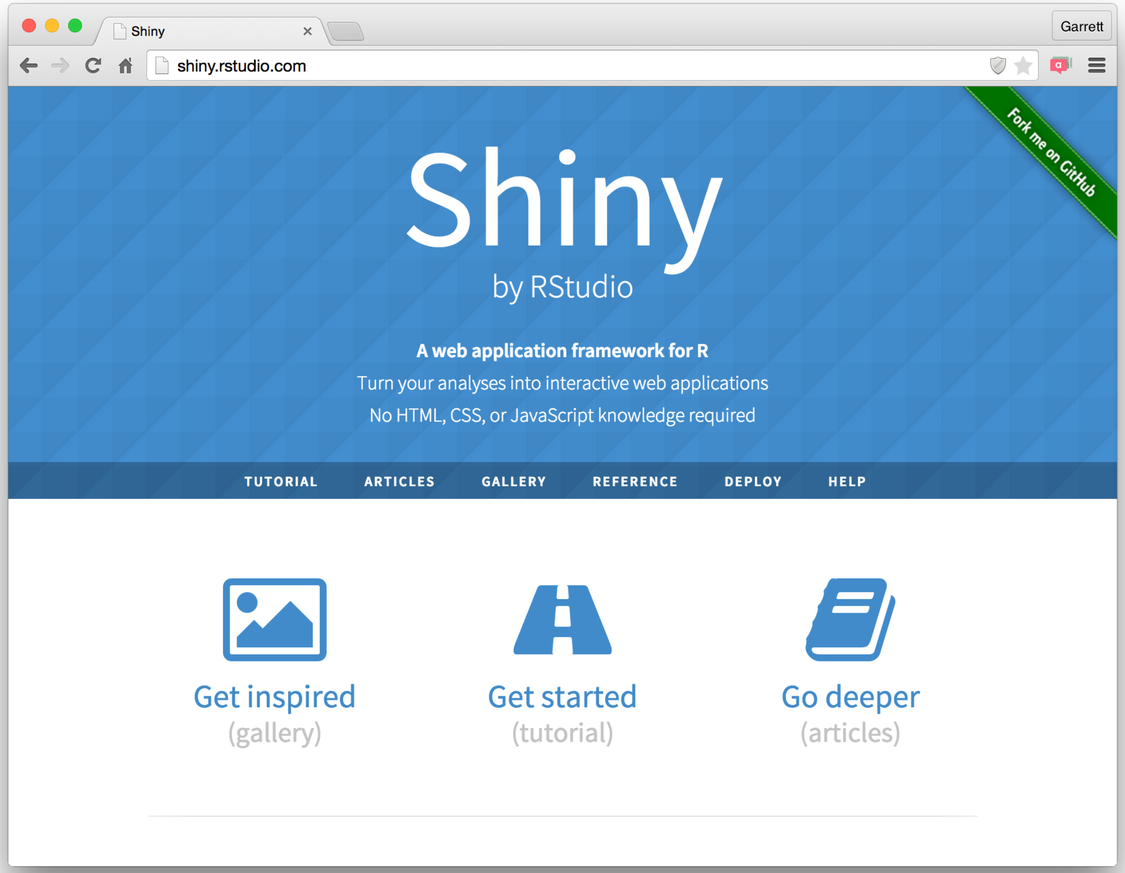
- **actionButton**(inputId, label, icon, ...)
- **actionLink**(inputId, label, icon, ...)
- **checkboxGroupInput**(inputId, label, choices, selected, inline)
- **checkboxInput**(inputId, label, value)
- **dateInput**(inputId, label, value, min, max, format, startview, weekstart, language)
- **dateRangeInput**(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
- **fileInput**(inputId, label, multiple, accept)
- **numericInput**(inputId, label, value, min, max, step)
- **passwordInput**(inputId, label, value)
- **radioButtons**(inputId, label, choices, selected, inline)
- **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)
- **sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)
- **submitButton**(text, icon) (Prevents reactions across entire app)
- **textInput**(inputId, label, value)

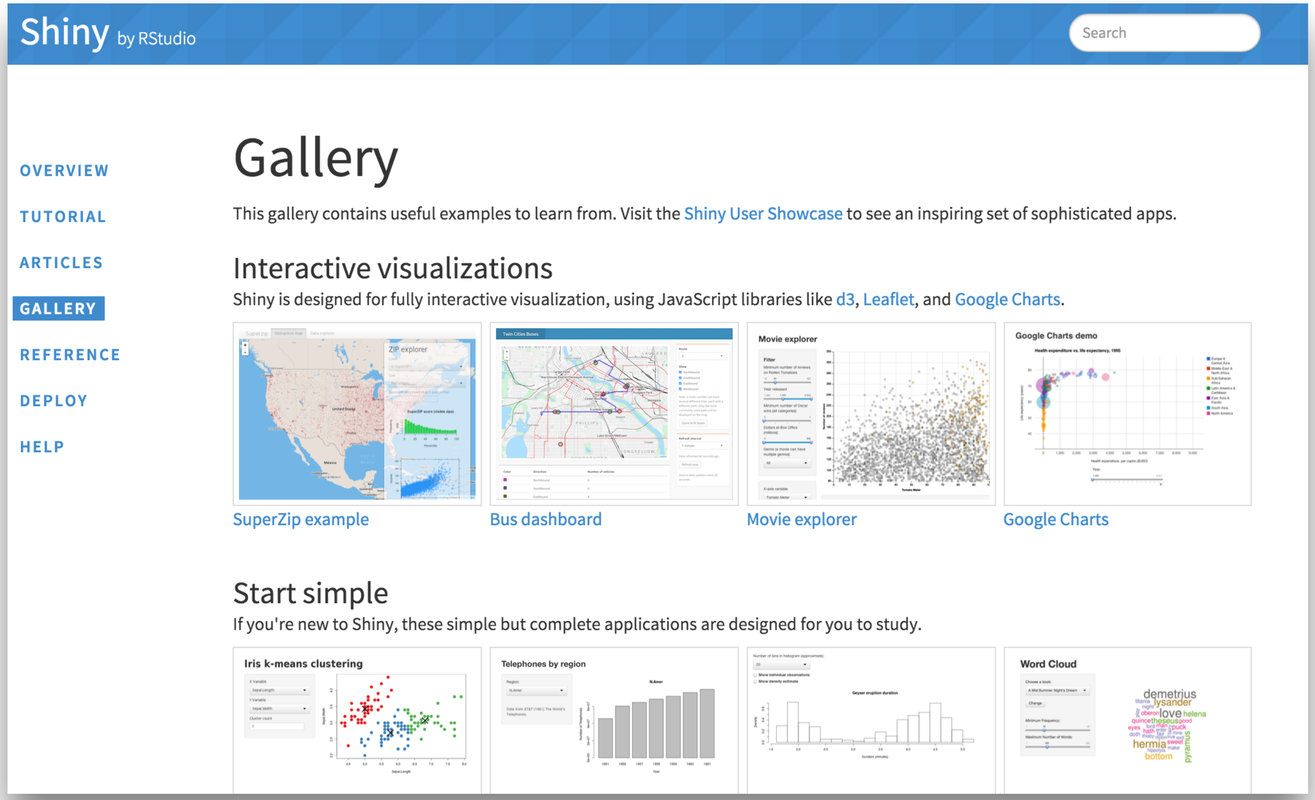
More cheat sheets at http://www.rstudio.com/resources/cheatsheets/

Learn more at shiny.rstudio.com/tutorial • shiny 0.12.0 • Updated: 6/15

The Shiny Development Center

shiny.rstudio.com





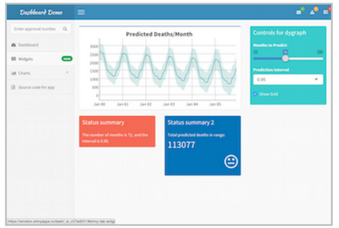
Shiny Showcase

www.rstudio.com/products/shiny/shiny-user-showcase/

RStudio

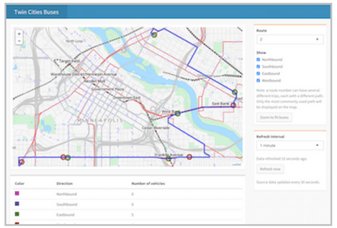
Products Resources Pricing About Us Blog

Shiny Apps for the Enterprise



Shiny Dashboard Demo

A dashboard built with Shiny.

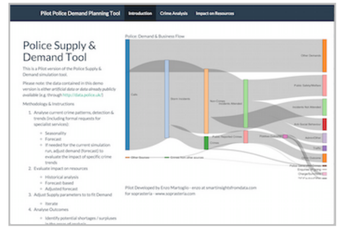


Location tracker

Track locations over time with streaming data.

Download monitor

Streaming download rates visualized as a bubble chart.



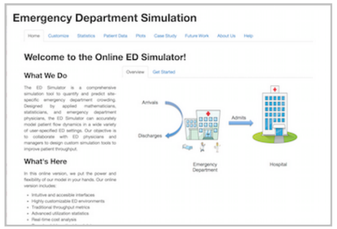
Supply and Demand

Forecast demand to plan resource allocation.

Industry Specific Shiny Apps

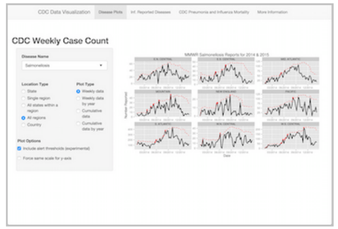
Economic Dashboard

Economic forecasting with macroeconomic indicators.



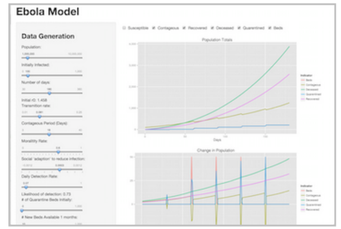
ER Optimization

An app that models patient flow.



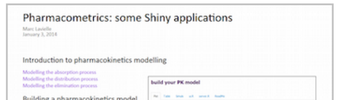
CDC Disease Monitor

Alert thresholds and automatic weekly updates.



Ebola Model

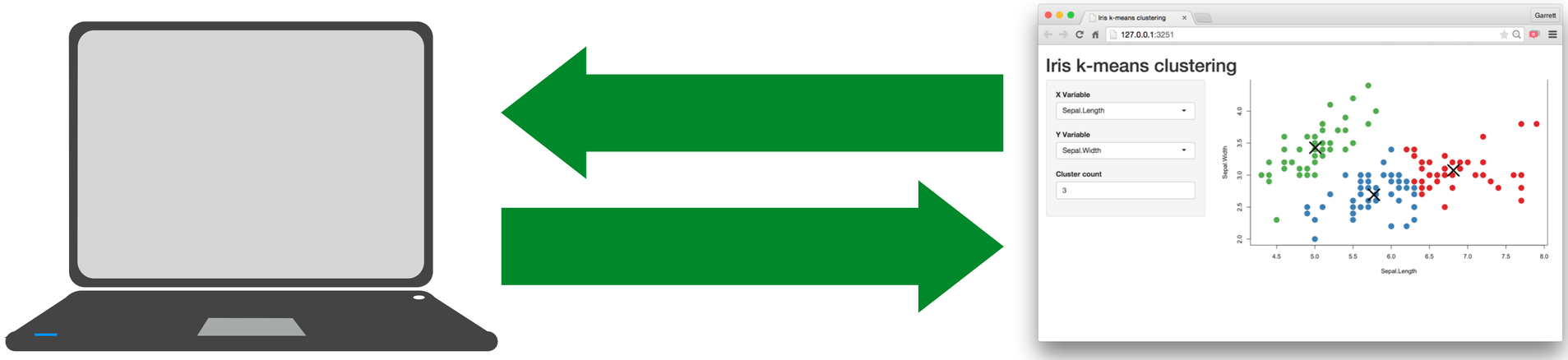
An epidemiological simulation.




HELLO
my name is
Garrett
Randy

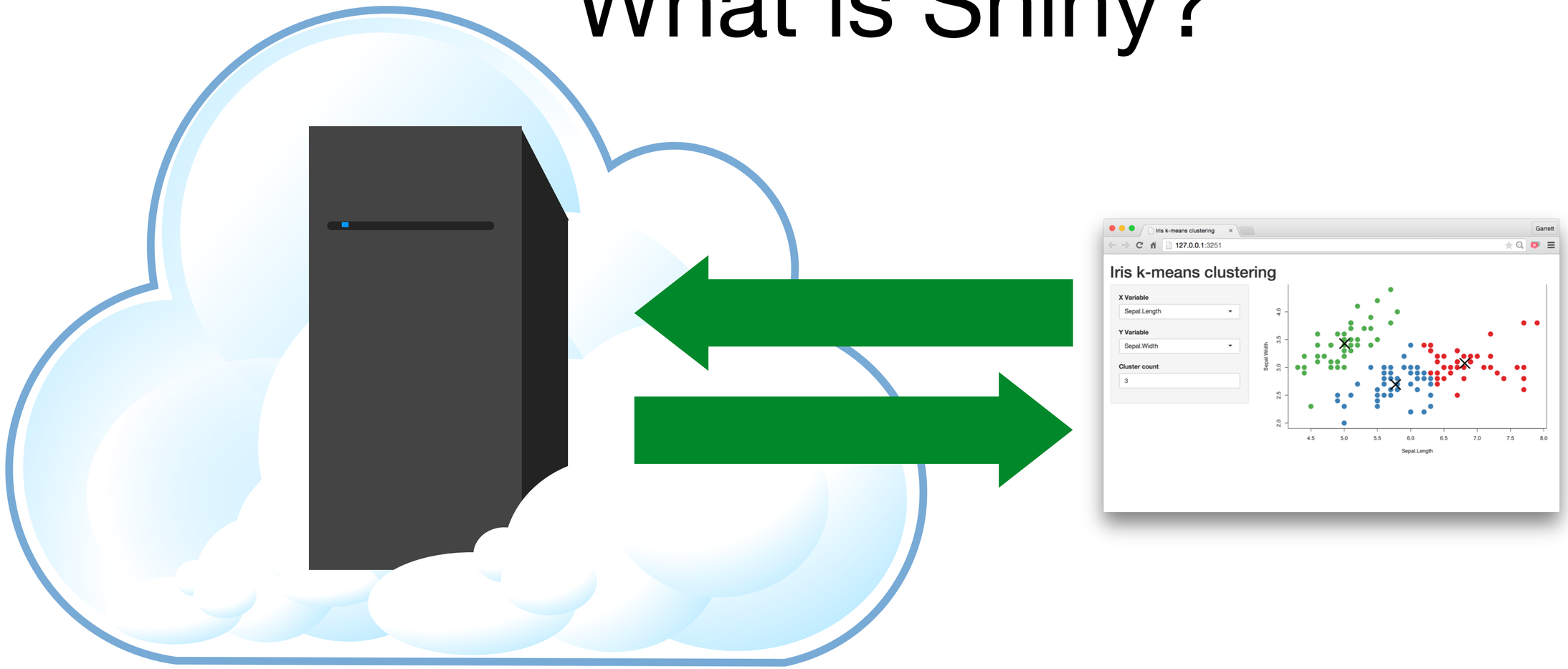
What is Shiny?

- shiny
  - R package that provides toolkit for creating shiny apps in R
  - `install.packages("shiny")`



Every Shiny app is maintained by a computer running R

What is Shiny?



Every Shiny app is maintained by a computer running R

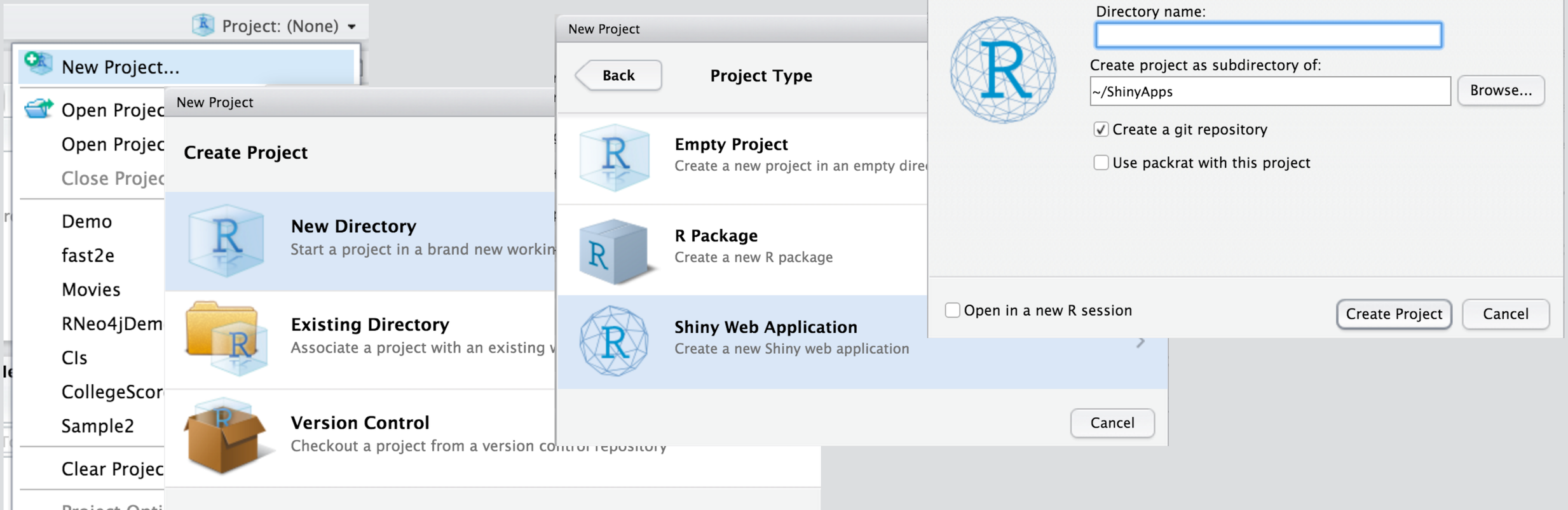
What is Shiny?

- shiny
  - R package that provides toolkit for creating shiny apps in R
  - `install.packages("shiny")`
- shiny-server
  - put apps on the web (free and pro versions available)
- shiny.io
  - RStudio can host your apps (free and pro accounts)

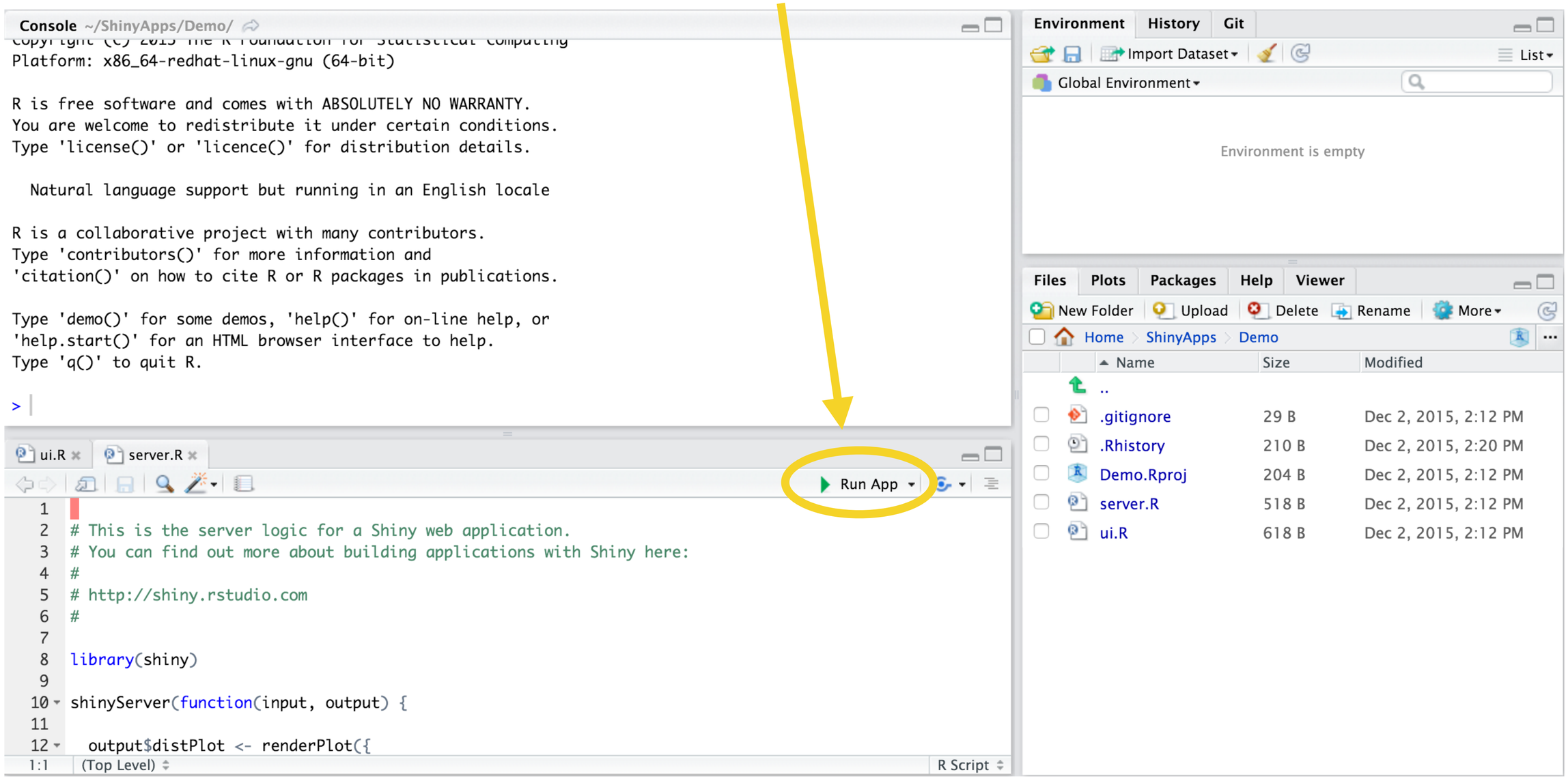
Your Turn

1. Log into the RStudio server and open a new R Script.
   http://rstudio.calvin.edu

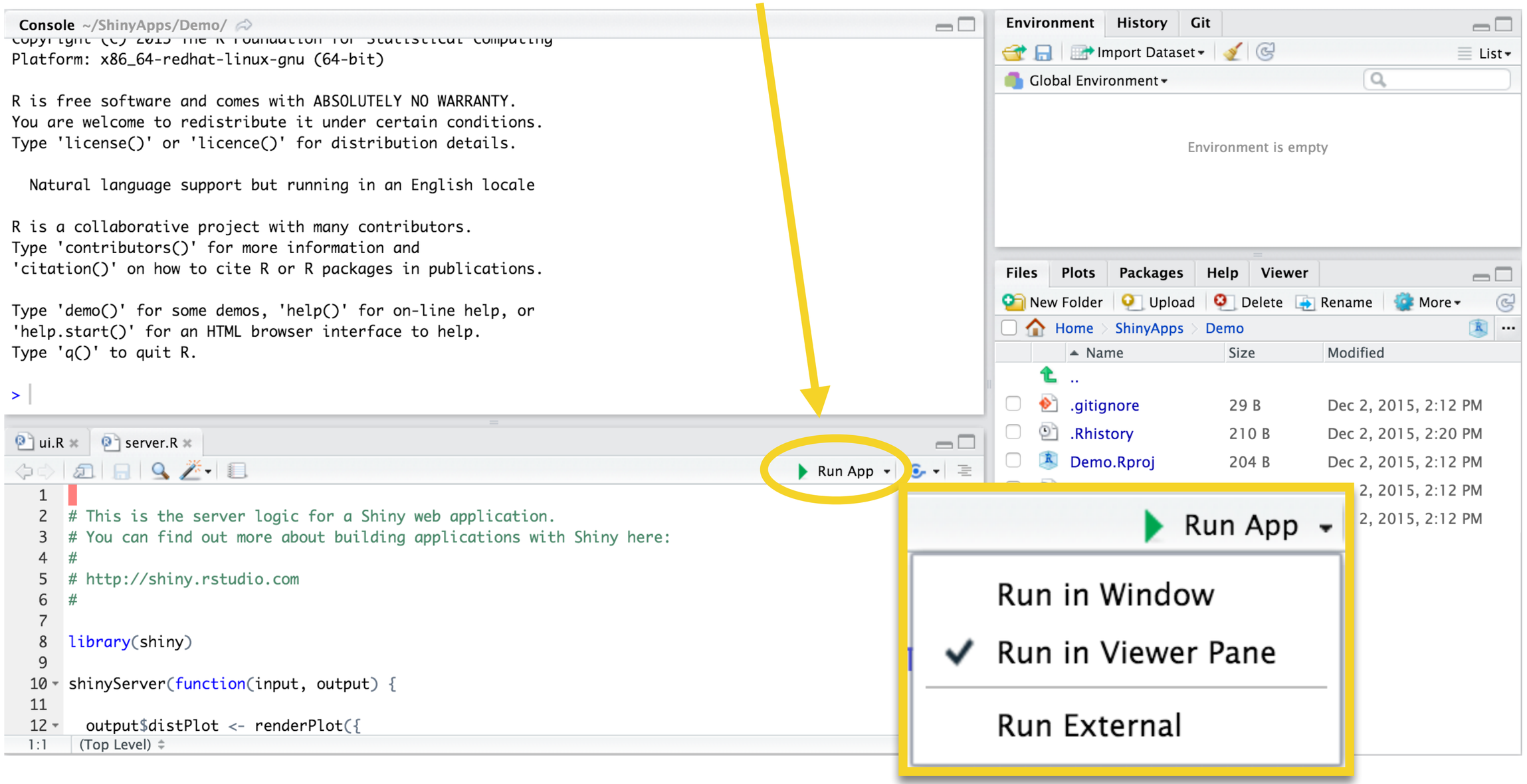
2. Create a **new Project**



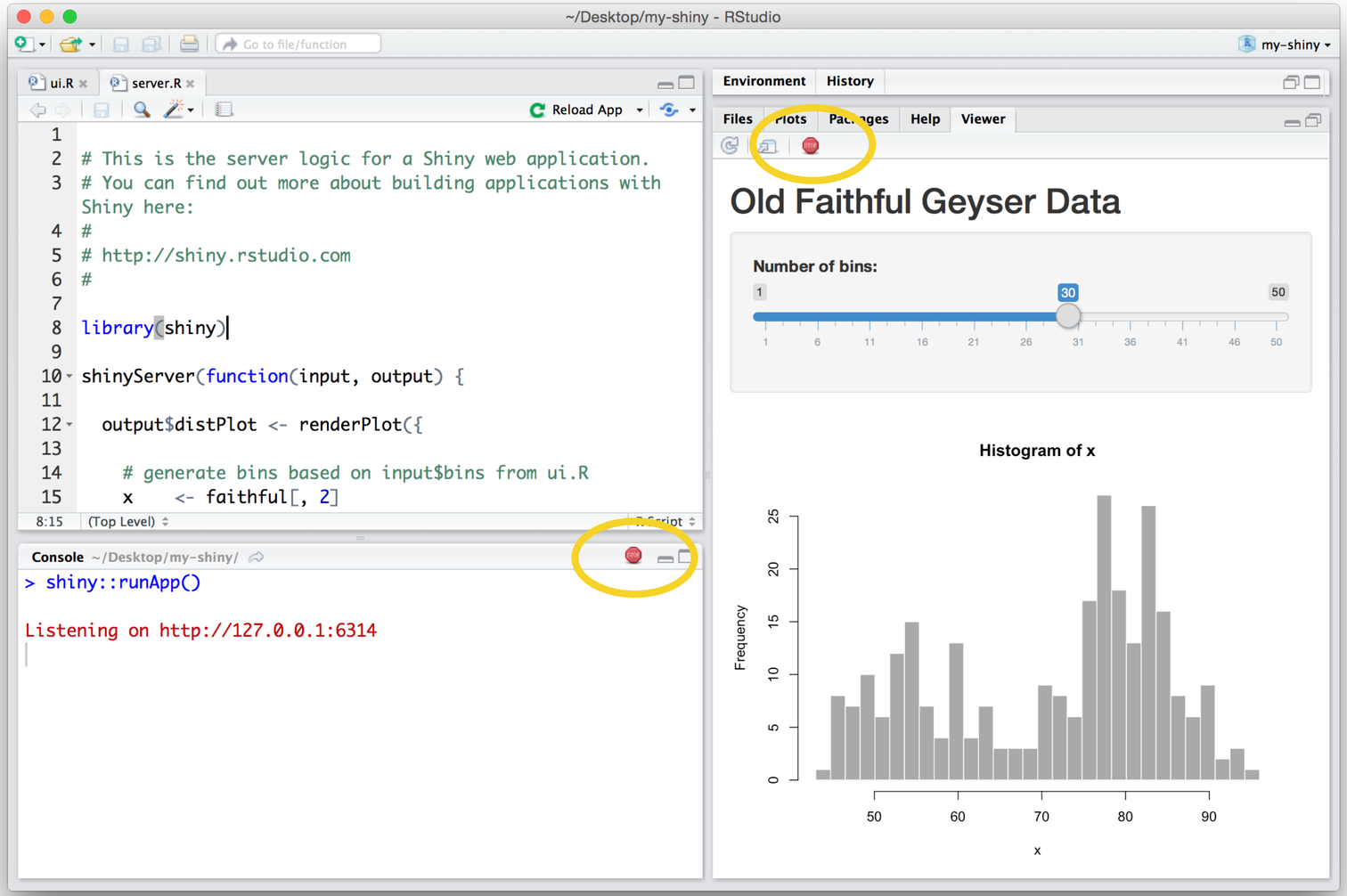
Run Your App


Console ~/ShinyApps/Demo/
Platform: x86_64-redhat-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
ui.R
server.R
Run App
# This is the server logic for a Shiny web application.
# You can find out more about building applications with Shiny here:
# http://shiny.rstudio.com
library(shiny)
shinyServer(function(input, output) {
output$distPlot <- renderPlot({
1:1
(Top Level)
R Script
Environment
History
Git
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
New Folder
Upload
Delete
Rename
More
Home
ShinyApps
Demo
Name
Size
Modified
..
.gitignore 29 B Dec 2, 2015, 2:12 PM
.Rhistory 210 B Dec 2, 2015, 2:20 PM
Demo.Rproj 204 B Dec 2, 2015, 2:12 PM
server.R 518 B Dec 2, 2015, 2:12 PM
ui.R 618 B Dec 2, 2015, 2:12 PM

Run Your App


Console ~/ShinyApps/Demo/
Platform: x86_64-redhat-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment History Git
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Home ShinyApps Demo
Name Size Modified
.gitignore 29 B Dec 2, 2015, 2:12 PM
.Rhistory 210 B Dec 2, 2015, 2:20 PM
Demo.Rproj 204 B Dec 2, 2015, 2:12 PM
ui.R
server.R
Run App
1 # This is the server logic for a Shiny web application.
2 # You can find out more about building applications with Shiny here:
3 #
4 # http://shiny.rstudio.com
5 #
6
7 library(shiny)
8
9 shinyServer(function(input, output) {
10
11 output$distPlot <- renderPlot({
12
1:1 (Top Level)
Run App
Run in Window
Run in Viewer Pane
Run External

Close your app



Outline

1. Components of an app

2. Reactivity

3. Interactive Plots

4. Sharing

5. Big Data

Components of an app

App template

The shortest viable shiny app

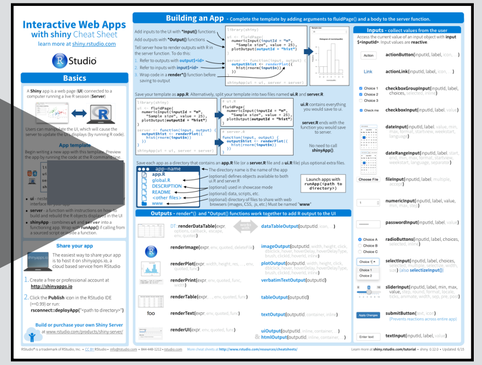
```
library(shiny)
ui <- fluidPage()


server <- function(input, output) {}


shinyApp(ui = ui, server = server)
```

Communication b/w UI and server

Starting from Scratch

1. Delete ui.R and server.R
2. Open a new R Script [File > New > RScript]
3. Write the code below in your R script and save as **app.R**
4. Hit Run App

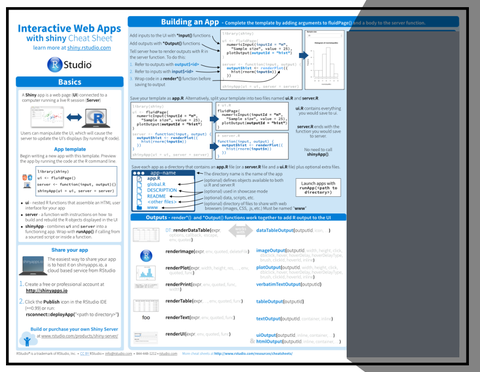
```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```



Add elements to your app as arguments to `fluidPage()`

```
library(shiny)
ui <- fluidPage("Hello, World")

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```

Inputs



Action — **actionButton**(inputId, label, icon, …)

Link — **actionLink**(inputId, label, icon, …)

Choice 1 / Choice 2 / Choice 3 — **checkboxGroupInput**(inputId, label, choices, selected, inline)

Check me — **checkboxInput**(inputId, label, value)

2015-06-08 — **dateInput**(inputId, label, value, min, max, format, startview, weekstart, language)

2015-06-08 to 2015-06-08 — **dateRangeInput**(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)

Choose File — **fileInput**(inputId, label, multiple, accept)

1 — **numericInput**(inputId, label, value, min, max, step)

•••••••• — **passwordInput**(inputId, label, value)

Choice A / Choice B / Choice C — **radioButtons**(inputId, label, choices, selected, inline)

Choice 1 (Choice 1, Choice 2) — **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)

0 5 10 (0 2 4 6 8 10) — **sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

Apply Changes — **submitButton**(text, icon)
(Prevents reactions across entire app)

Enter text — **textInput**(inputId, label, value)

Inputs

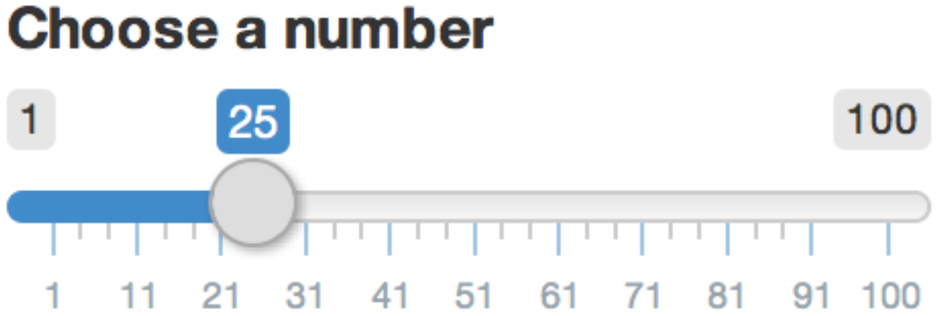
collect a value from your user.

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", “Choose a number”, 1, 100, 25)
)
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

unique Id

displayed text

Syntax




```
sliderInput(inputId = "num", label = “Choose a number", …)
```

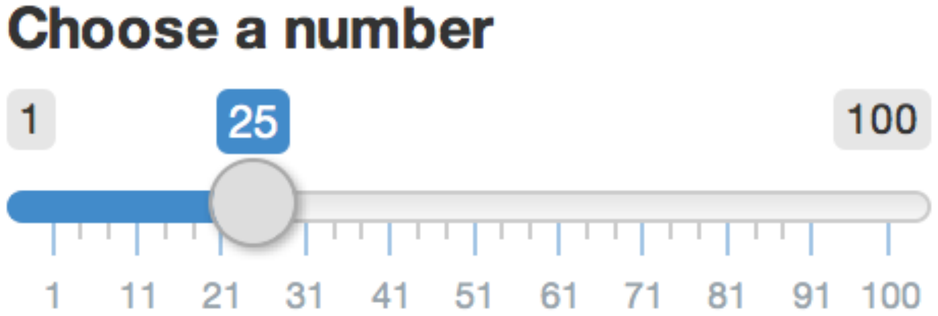
Notice:
Id not ID

input name
(used by server)

input specific
arguments

```
?sliderInput
```

Syntax




```
sliderInput(inputId = "num", label = “Choose a number", …)
```

Notice: Id not ID

input name (used by server)

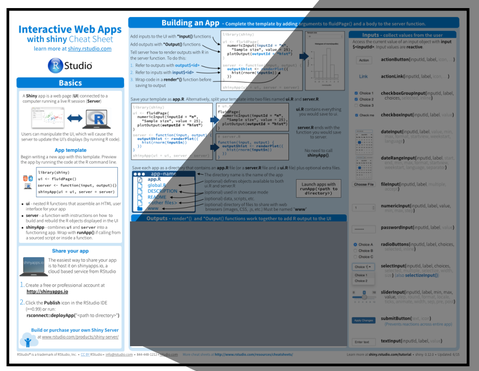
label to display

input specific arguments

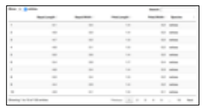
```
?sliderInput
```

Outputs

display output from R.



Outputs - render*() and *Output() functions work together to add R output to the UI



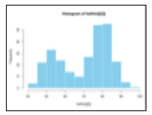
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)

works with

dataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)



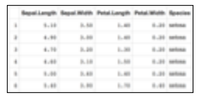
renderPlot(expr, width, height, res, …, env, quoted, func)

plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

```
'data.frame':  3 obs. of  2 variables:
 $ Sepal.Length: num  5.1 4.9 4.7
 $ Sepal.Width : num  3.5 3 3.2
```

renderPrint(expr, env, quoted, func, width)

verbatimTextOutput(outputId)



renderTable(expr,…, env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, …)
& **htmlOutput**(outputId, inline, container, …)

Outputs

display output from R.

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 10, 5),
)

server <- function(input, output) {}




shinyApp(ui = ui, server = server)
```

Build outputs in 3 steps:

Outputs

display output from R.

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("bar")
)
server <- function(input, output) {}




shinyApp(ui = ui, server = server)
```

Build outputs in 3 steps:

1. Add a *Output() function to ui (places output)

*Output()

To display output, add it to `fluidPage()` with an `*Output()` function

```
plotOutput(outputId = "bar")
```

*Output()

To display output, add it to `fluidPage()` with an `*Output()` function

```
plotOutput(outputId = "bar")
```

the type of output to display

*Output()

To display output, add it to `fluidPage()` with an `*Output()` function

```
plotOutput(outputId = "bar")
```

the type of output to display

name to give to the output object

Outputs

display output from R.

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("bar")
)
server <- function(input, output) {
  renderPlot({
     barplot(50, ylim = c(0, 100))
  })
}
shinyApp(ui = ui, server = server)
```

Build outputs in 3 steps:

1. Add a *Output() function to ui (places output)

2. Make with render*() function in server (builds output)

render*()

Builds reactive output to display in UI

```
renderPlot({ barplot(50, ylim = c(0, 100)) })
```

render*()

Builds reactive output to display in UI

```
renderPlot({ barplot(50, ylim = c(0, 100)) })
```

type of object to build

render*()

Builds reactive output to display in UI

```
renderPlot({ barplot(50, ylim = c(0, 100)) })
```

type of object to build

code block that builds the object

Outputs

display output from R.

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("bar")
)
server <- function(input, output) {
  output$bar <- renderPlot({
     barplot(50, ylim = c(0, 100))
  })
}
shinyApp(ui = ui, server = server)
```

Build outputs in 3 steps:

1. Add a *Output() function to ui (places output)
2. Make with render*() function in server (builds output)
3. Save to output$ list (stores output)

Outputs

display output from R.

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("bar")
)
server <- function(input, output) {
  output$bar <- renderPlot({
     barplot(50, ylim = c(0, 100))
  })
}
shinyApp(ui = ui, server = server)
```

Match names

Build outputs in 3 steps:

1. Add a *Output() function to ui (places output)
2. Make with render*() function in server (builds output)
3. Save to output$ list (stores output)

Your Turn

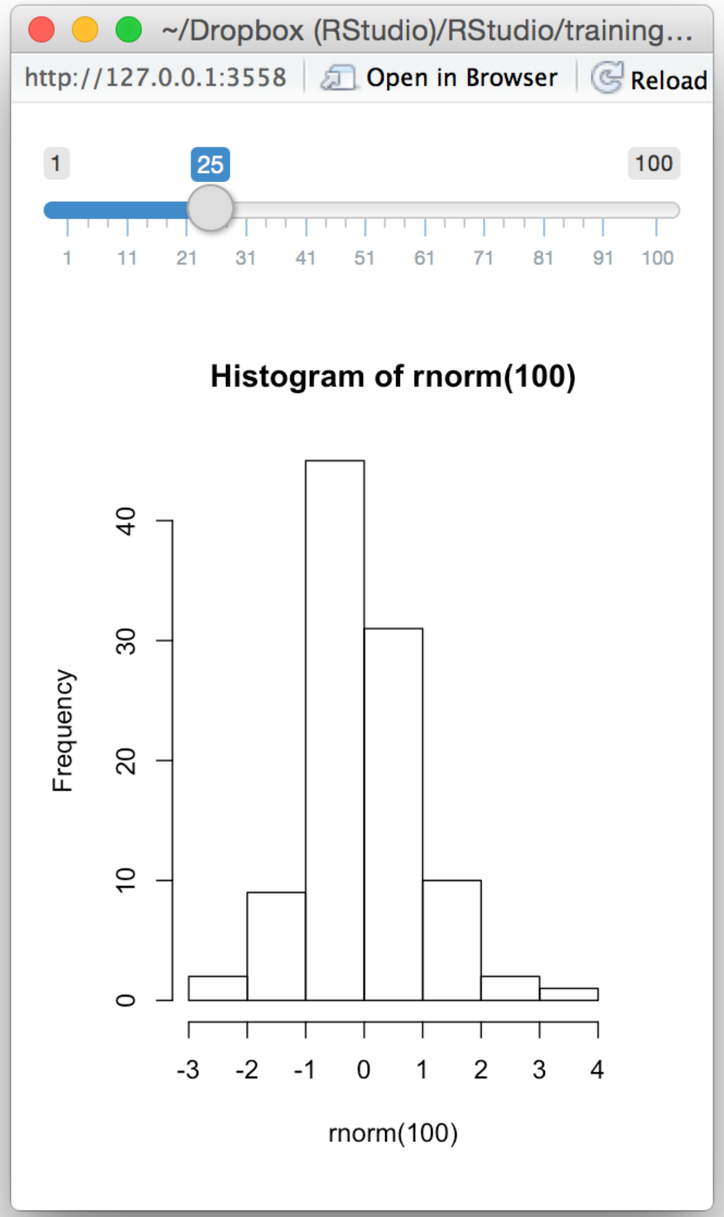
Make a new app that contains:

1. A slider that goes from 1 to 100
2. A histogram 100 random normal values

```
hist(rnorm(100))              # base
histogram( ~ rnorm(100)) # lattice
qplot(rnorm(100))             # ggplot2
```

```
library(shiny)
ui <- fluidPage(
   sliderInput("num", "", 1, 100, 25),
   plotOutput("hist")
)
server <- function(input, output) {
   output$hist <- renderPlot({
      hist(rnorm(100))
   })
}
shinyApp(ui = ui, server = server)
```

To do: add interaction between slider and plot



App template

The shortest viable shiny app

```
library(shiny)
ui <- fluidPage()


server <- function(input, output) {}


shinyApp(ui = ui, server = server)
```

Communication b/w UI and server

Reactions

The **input$** list stores the current value of each input object under its name.

```
sliderInput(inputId = "num", …)
```

```
input$num
```

Reactions

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("bar")
)
server <- function(input, output) {
  output$bar <- renderPlot({
   barplot(input$num, ylim=c(0, 100))
  })
}
shinyApp(ui = ui, server = server)
```

Shiny will update an output whenever an input value changes *if the output uses the input value in its render function*.

An input value

Reactions

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("bar")
)
server <- function(input, output) {
  output$bar <- renderPlot({
    input$num
    barplot(50, ylim=c(0,100))
  })}

shinyApp(ui = ui, server = server)
```

Shiny will update an output whenever an input value changes *if the output uses the input value in its render function*.

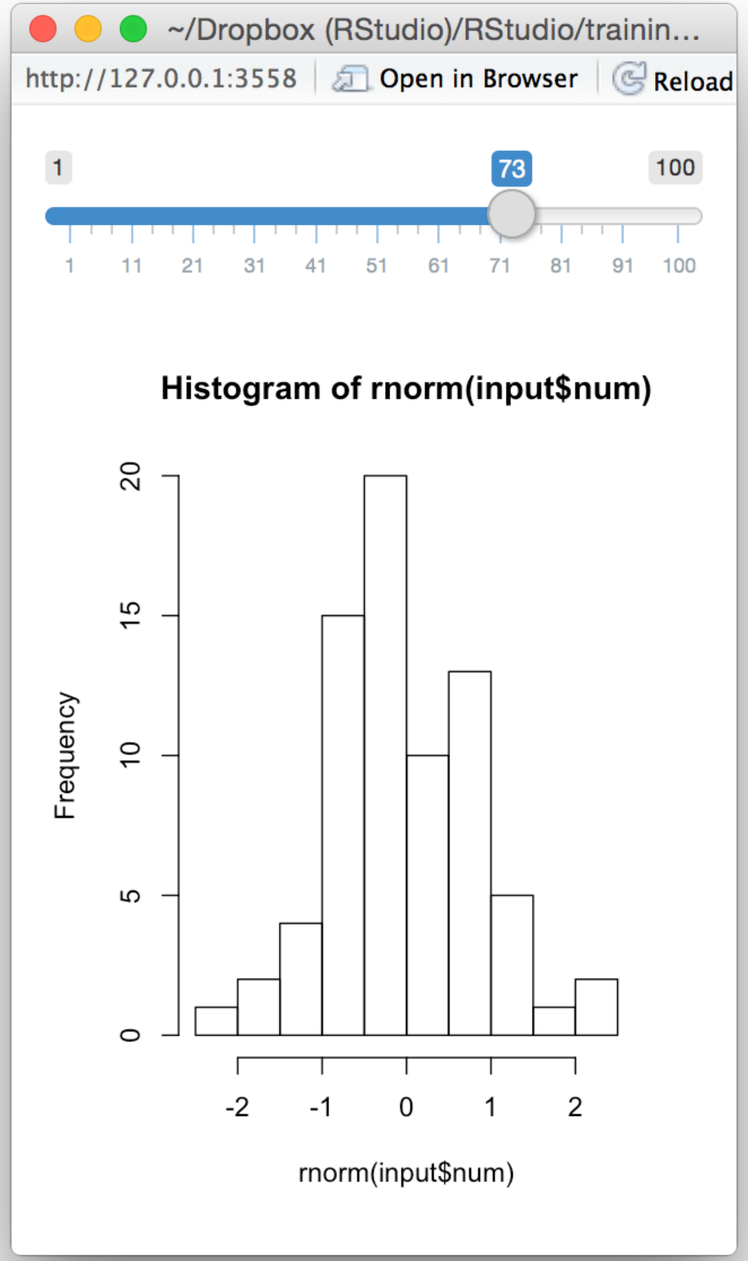
An input value

Your Turn

Change your app to make the number of random normal values in the histogram react to the value of the slider.

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("bar")
)
server <- function(input, output) {
  output$bar <- renderPlot({
   barplot(input$num, ylim=c(0, 100))
  })
}
shinyApp(ui = ui, server = server)
```

```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
shinyApp(ui = ui, server = server)
```



Recap

Recap

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template

Recap

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template

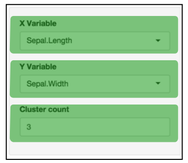
Add elements as arguments to **fluidPage()**

Recap

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template

Add elements as arguments to **fluidPage()**



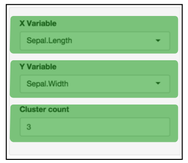
Create reactive inputs with an ***Input()** function

Recap

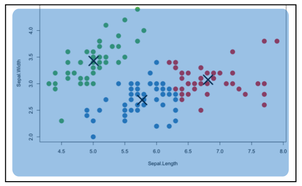
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template

Add elements as arguments to **fluidPage()**



Create reactive inputs with an ***Input()** function



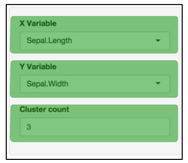
Display R results with an ***Output()** function

Recap

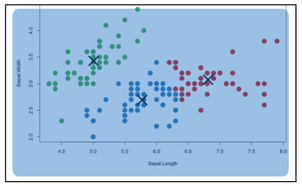
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template

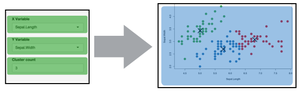
Add elements as arguments to **fluidPage()**



Create reactive inputs with an ***Input()** function



Display R results with an ***Output()** function



Use the server function to assemble inputs into outputs

Recap: Server

Recap: Server

Use the server function to assemble inputs into outputs. Follow 3 rules:

Recap: Server

Use the server function to assemble inputs into outputs. Follow 3 rules:

```
output$hist <-
```

1. Save the output that you build to **output$**

Recap: Server

Use the server function to assemble inputs into outputs. Follow 3 rules:

`output$hist <-`

1. Save the output that you build to **output$**

```
renderPlot({
  hist(rnorm(input$num))
})
```

2. Build the output with a **render*()** function

Recap: Server

Use the server function to assemble inputs into outputs. Follow 3 rules:

`output$hist <-`

1. Save the output that you build to **output$**

```
renderPlot({
  hist(rnorm(input$num))
})
```

2. Build the output with a **render*()** function

`input$num`

3. Access input values with **input$**

Recap: Server

Use the server function to assemble inputs into outputs. Follow 3 rules:

```
output$hist <-
```

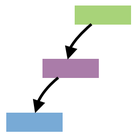
1. Save the output that you build to **output$**

```
renderPlot({
  hist(rnorm(input$num))
})
```

2. Build the output with a **render*()** function

```
input$num
```

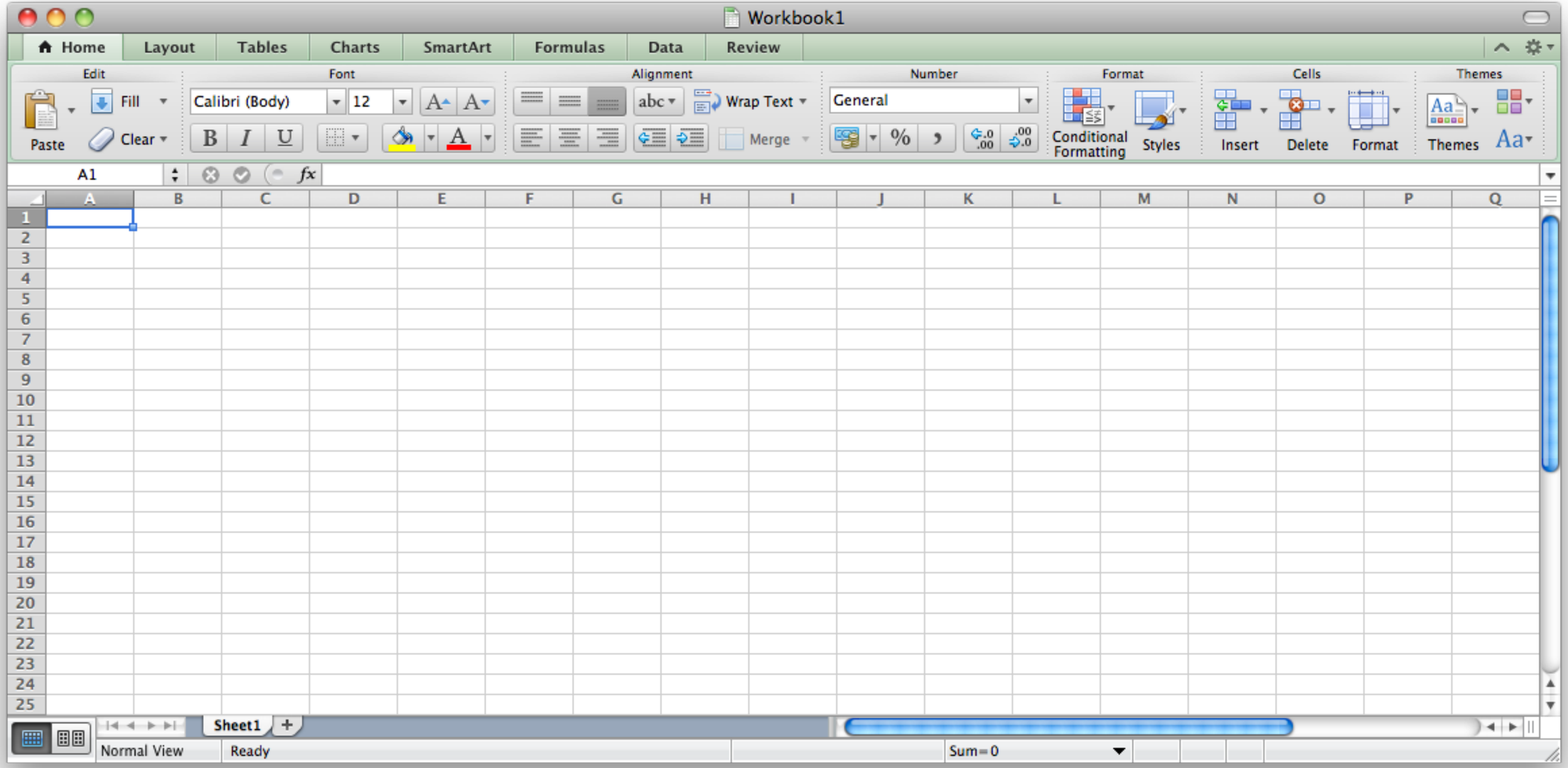
3. Access input values with **input$**

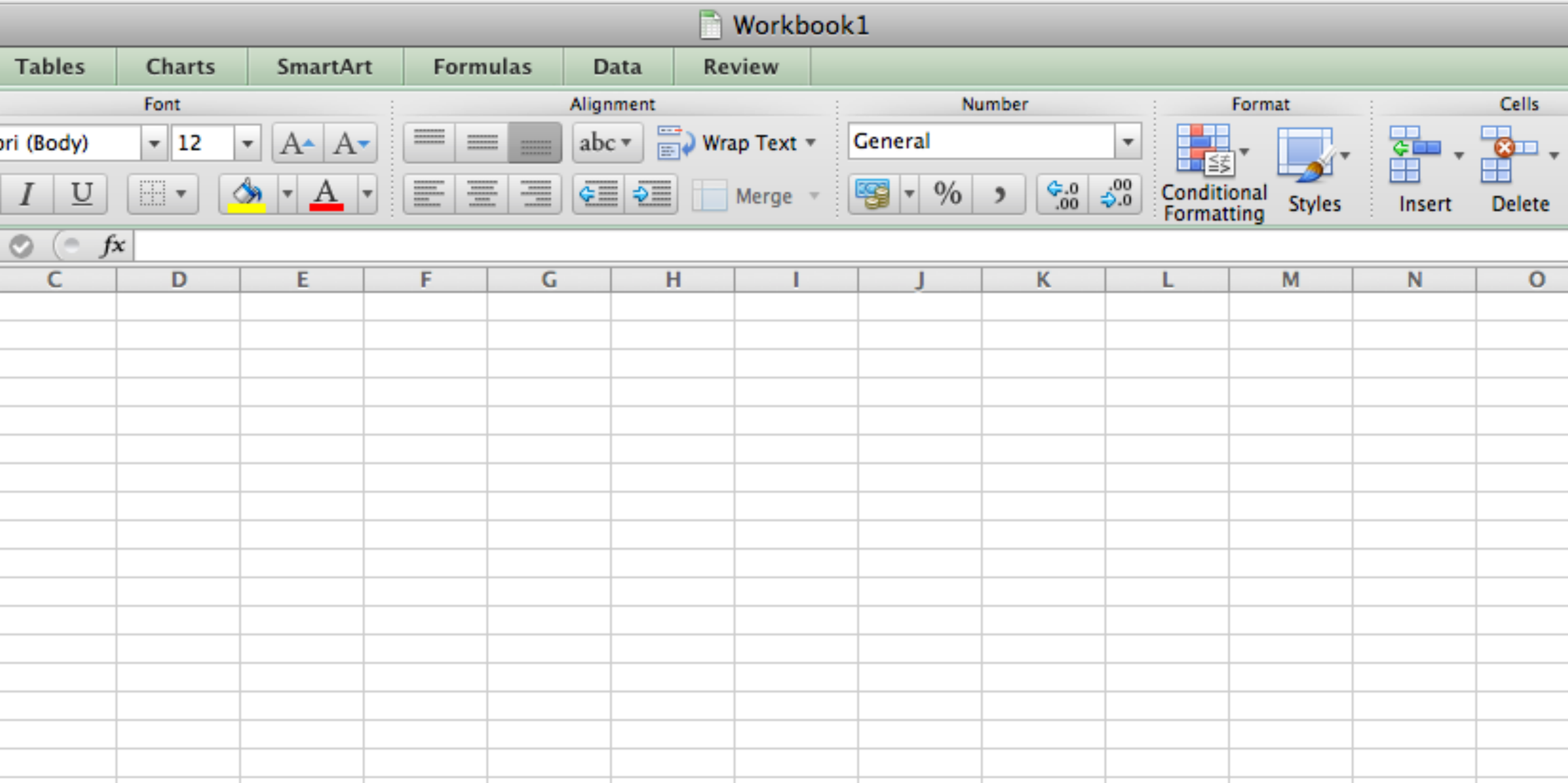


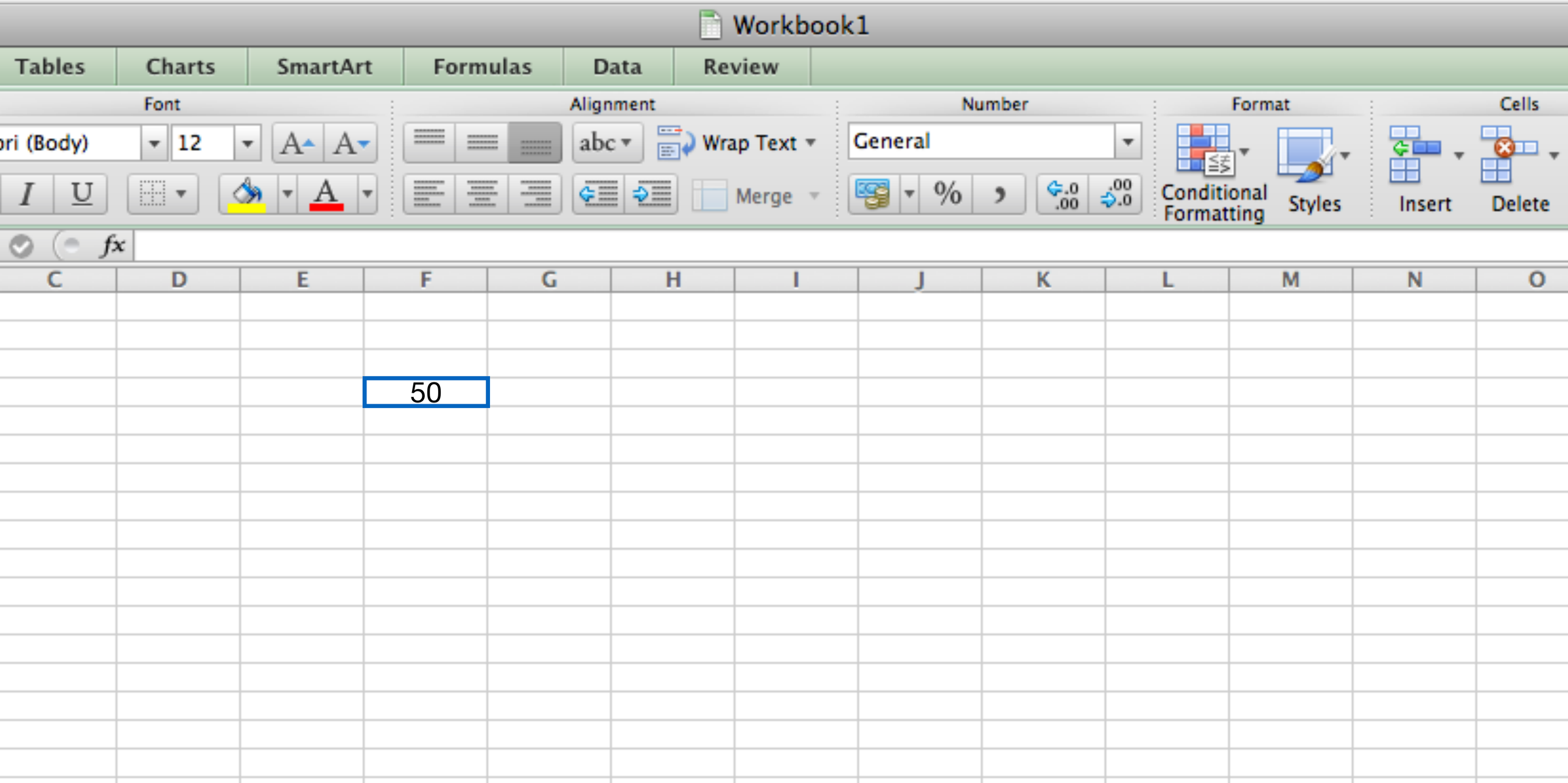
Create reactivity by using **Inputs** to build **rendered Outputs**

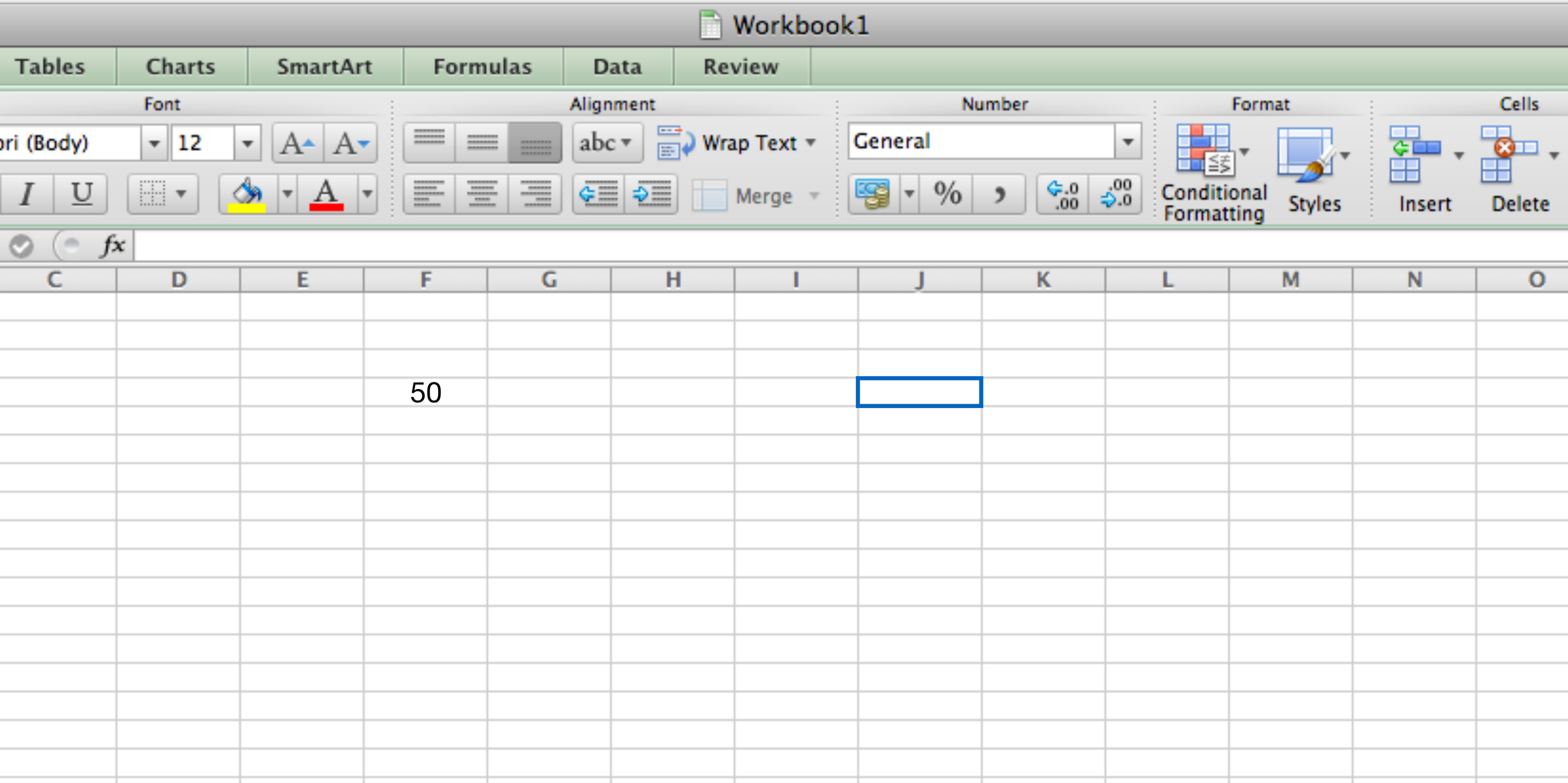
Reactivity

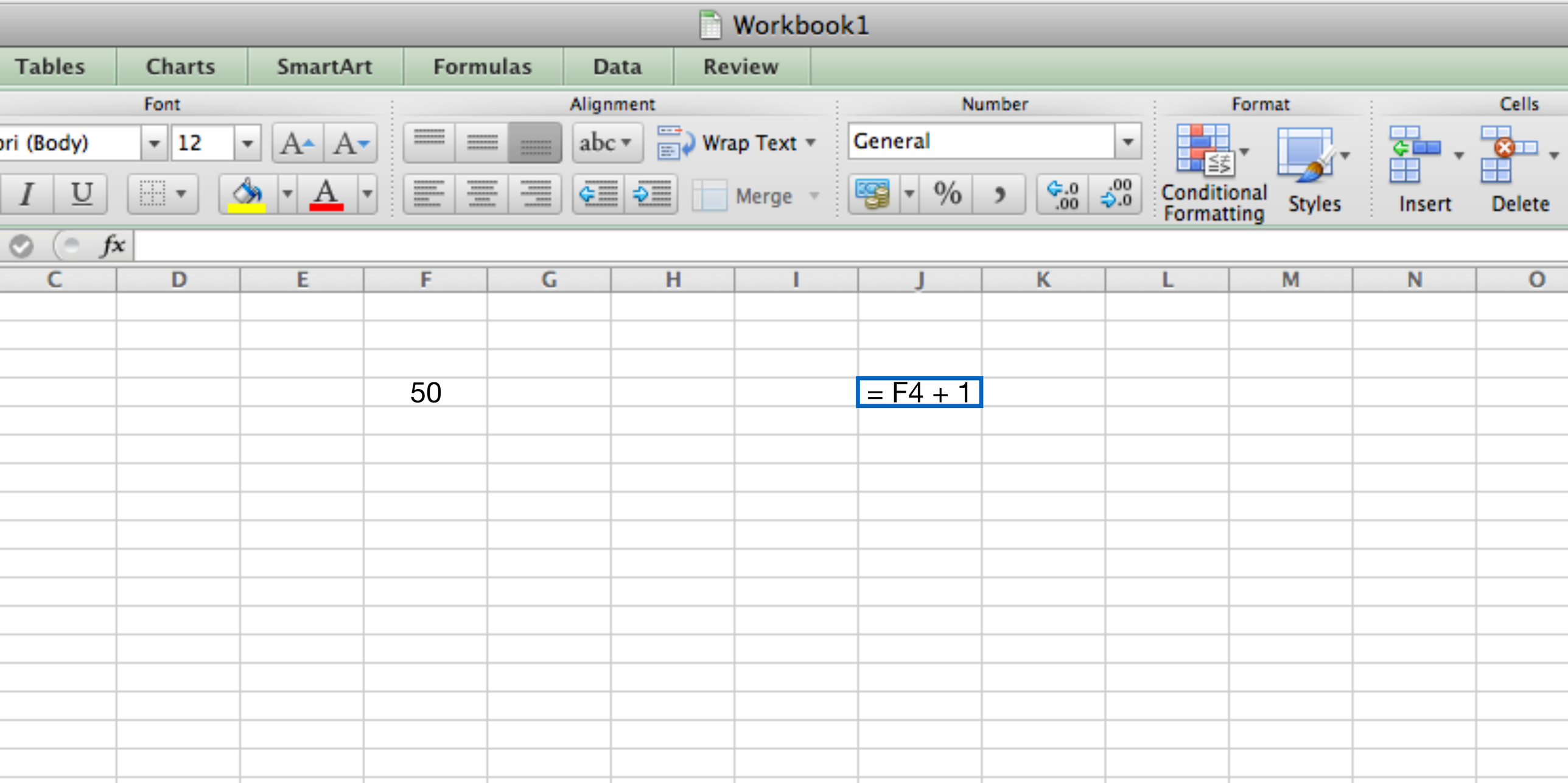
Think Excel.

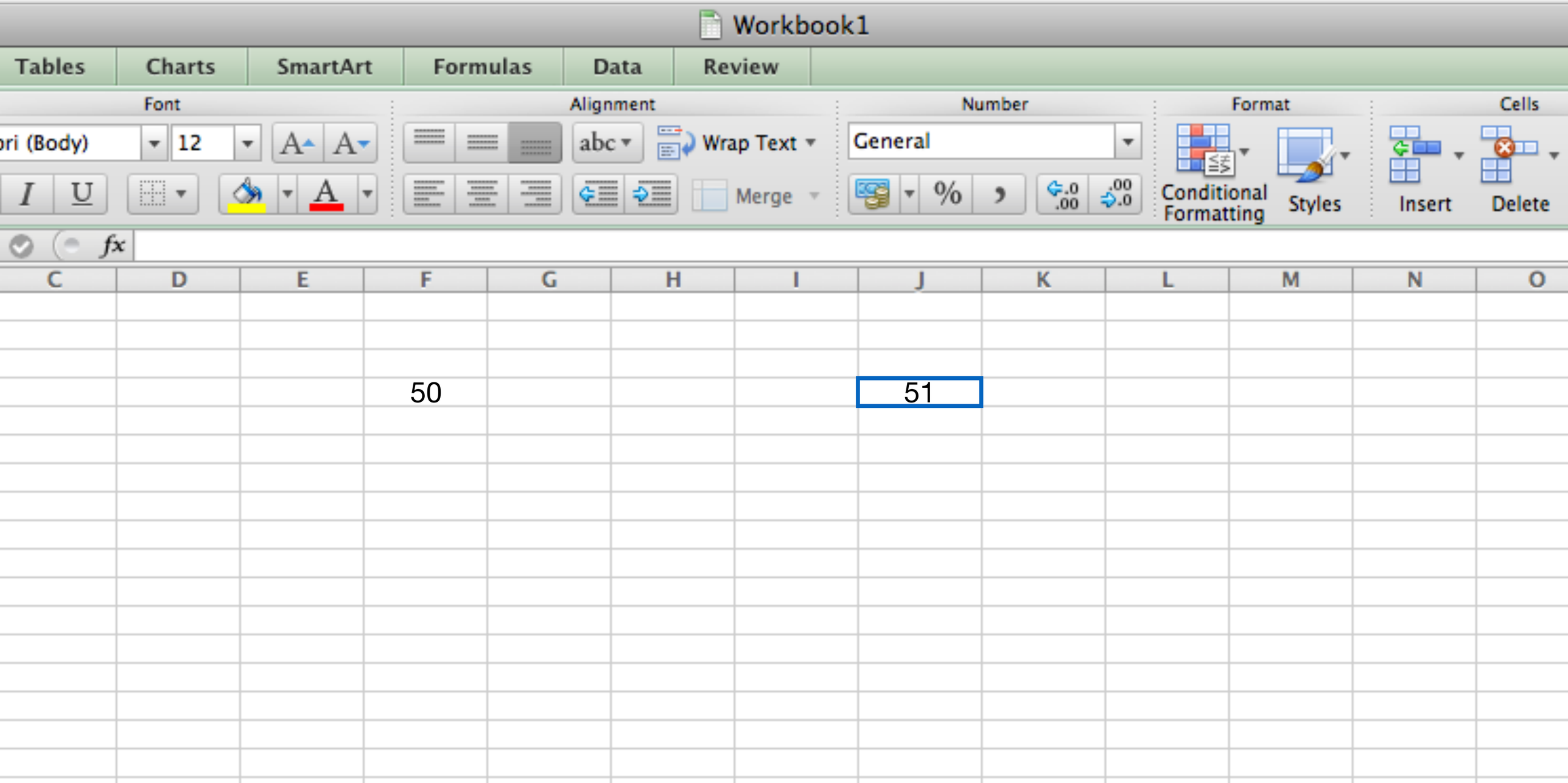

Workbook1
Home
Layout
Tables
Charts
SmartArt
Formulas
Data
Review
Edit
Font
Alignment
Number
Format
Cells
Themes
Paste
Fill
Clear
Calibri (Body)
12
Wrap Text
Merge
General
Conditional Formatting
Styles
Insert
Delete
Format
Themes
A1
Sheet1
Normal View
Ready
Sum=0

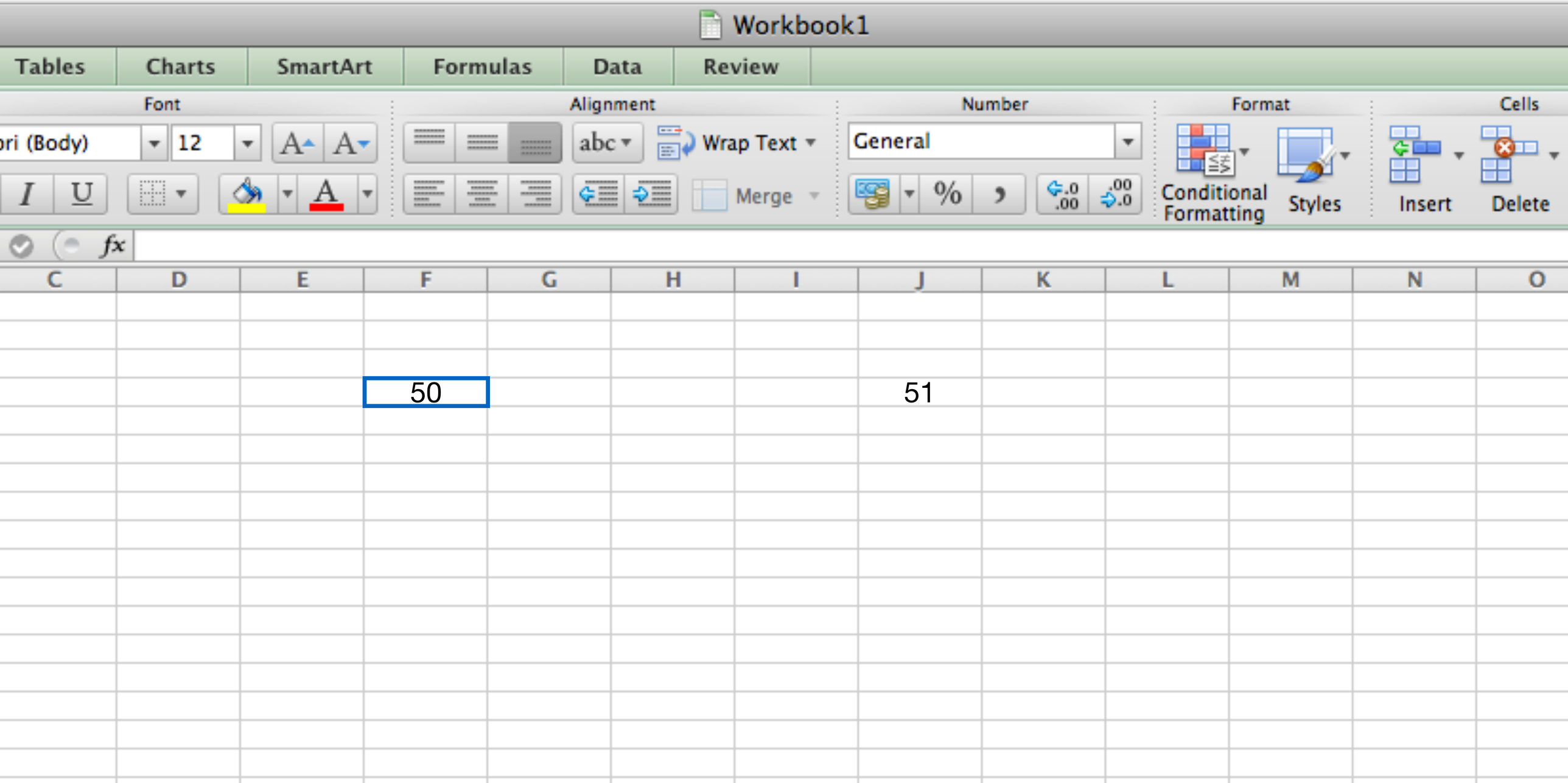

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
fx
C
D
E
F
G
H
I
J
K
L
M
N
O

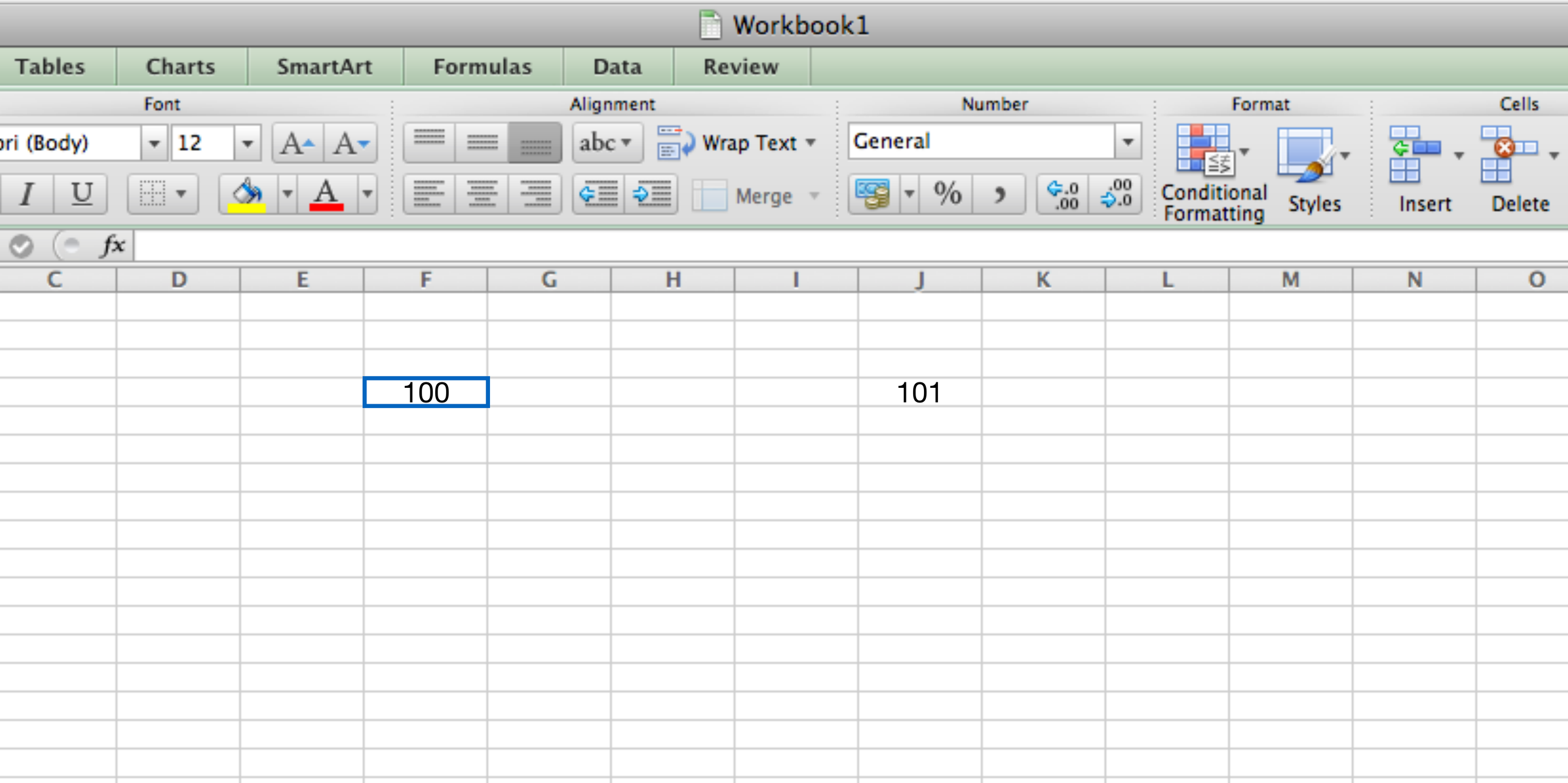

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
50
C
D
E
F
G
H
I
J
K
L
M
N
O

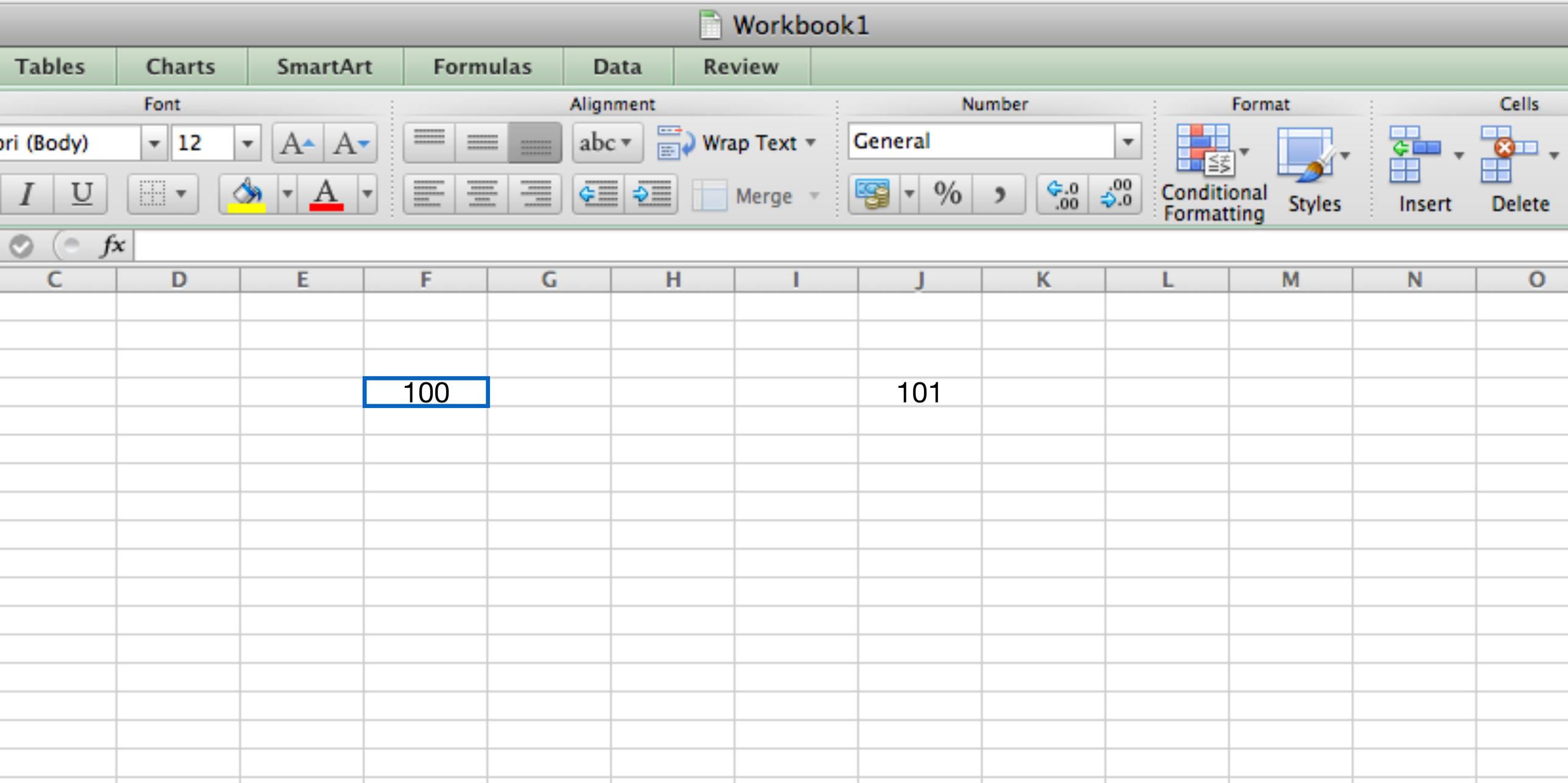

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
fx
C
D
E
F
G
H
I
J
K
L
M
N
O
50

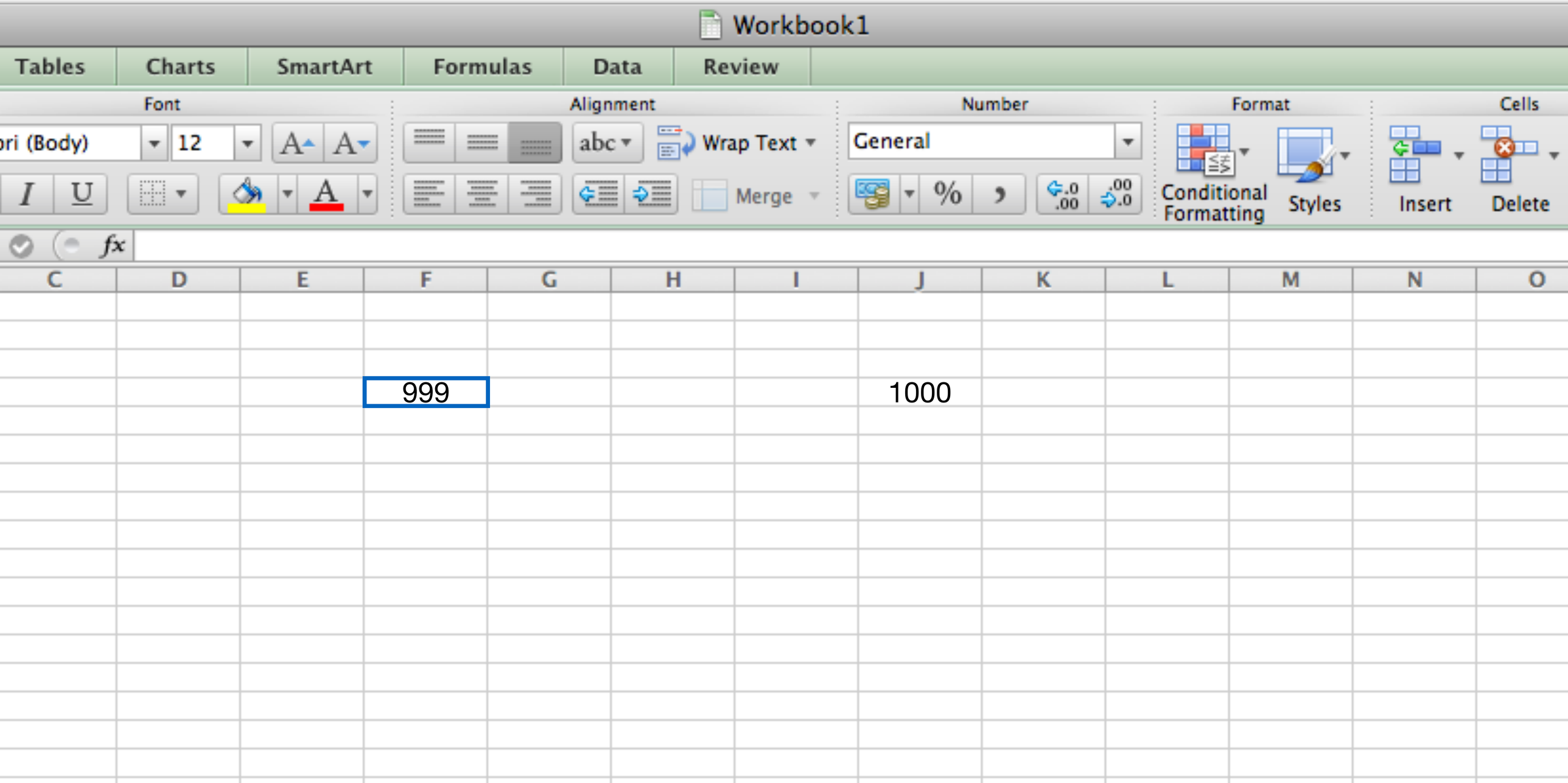

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
C
D
E
F
G
H
I
J
K
L
M
N
O
50
= F4 + 1

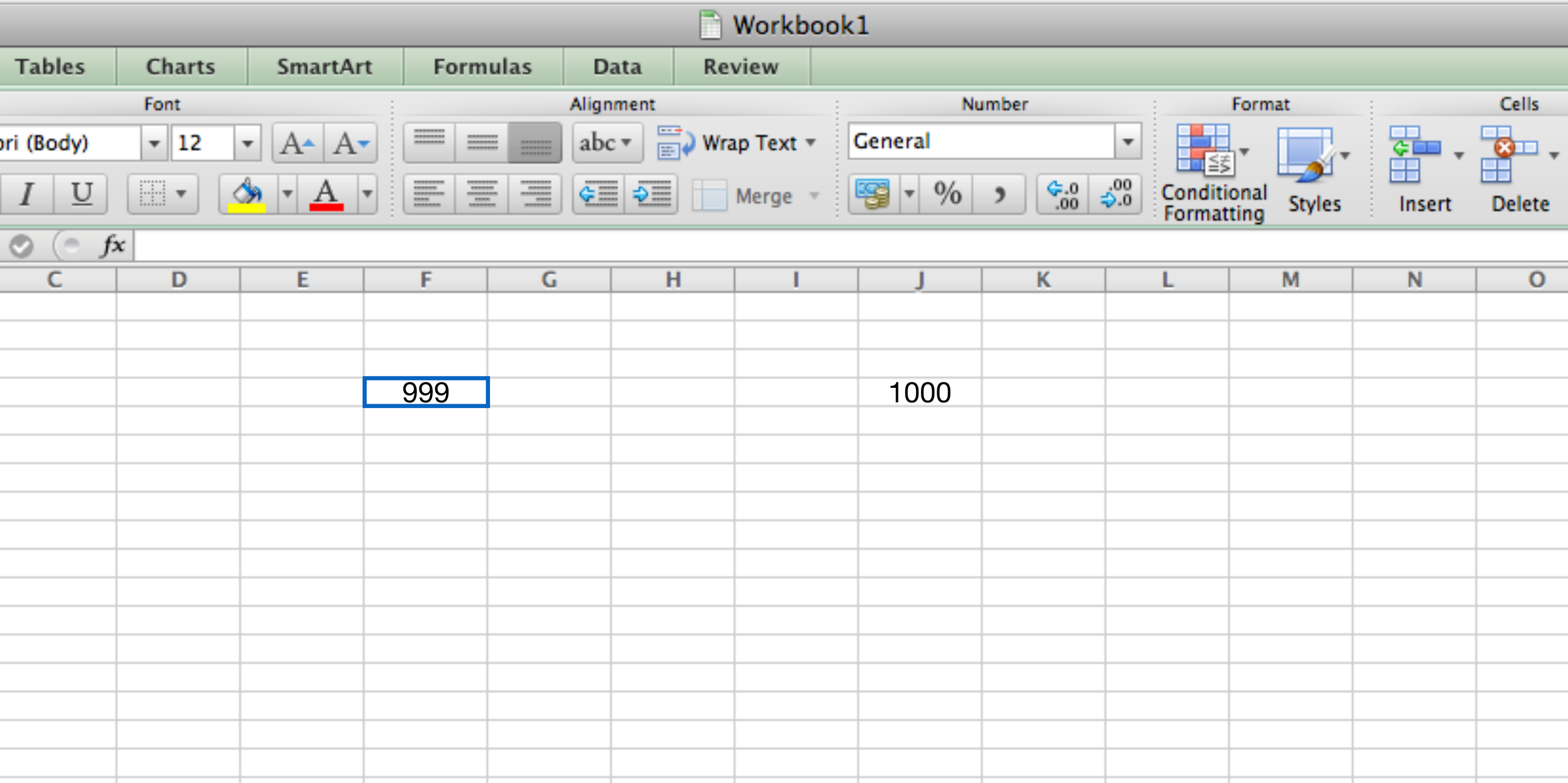

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
C
D
E
F
G
H
I
J
K
L
M
N
O
50
51

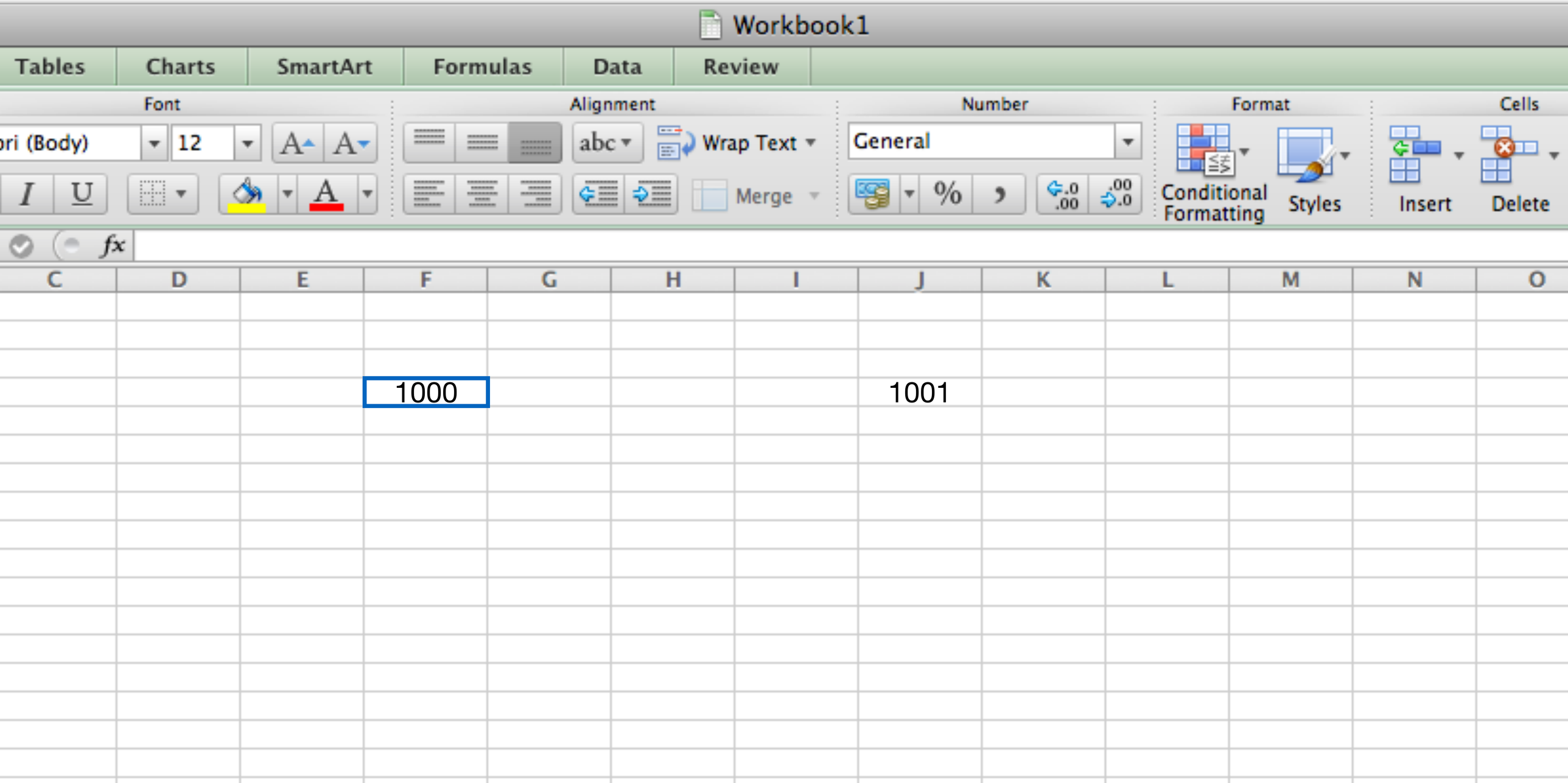

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
C
D
E
F
G
H
I
J
K
L
M
N
O
50
51

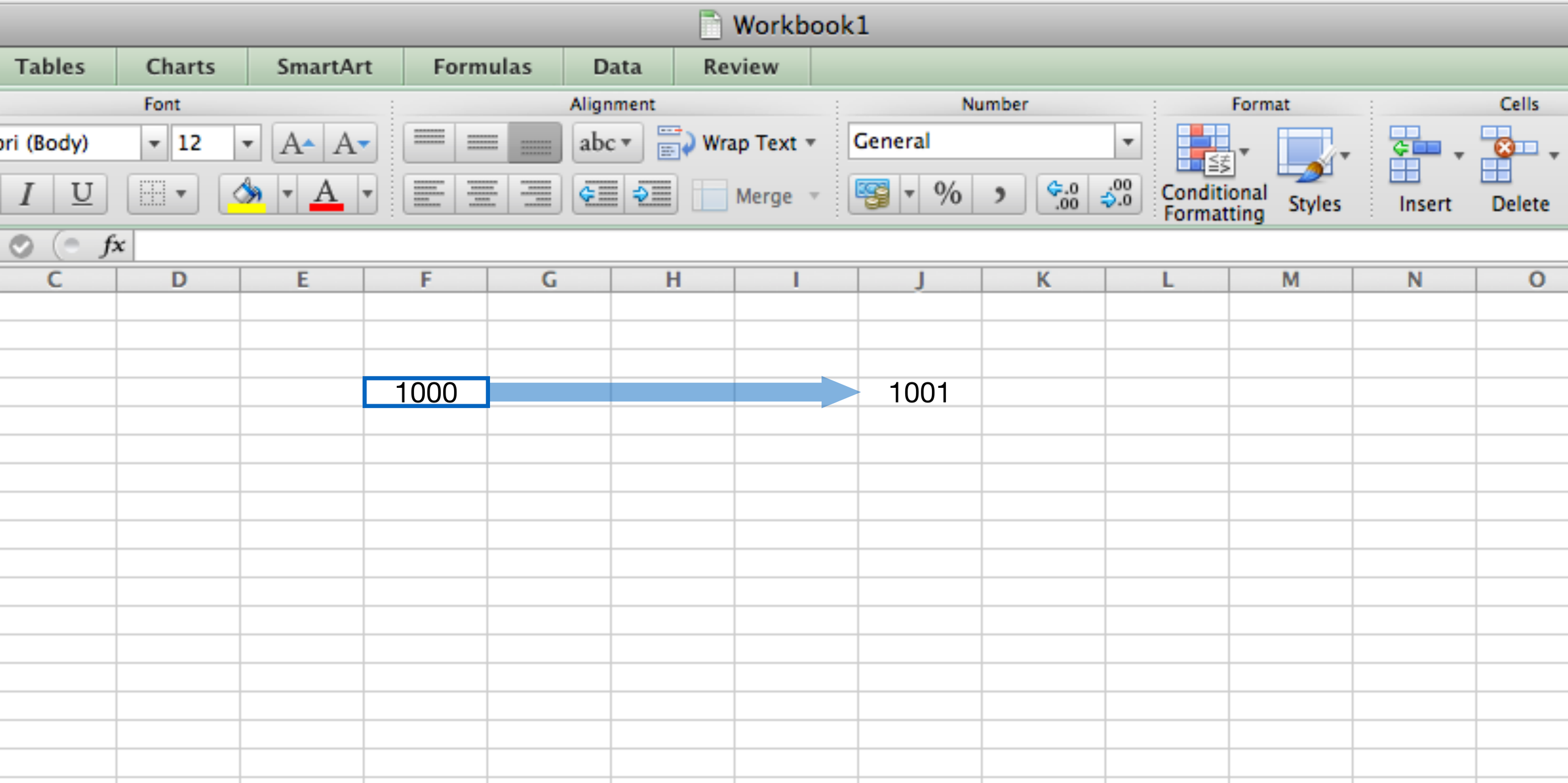

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
100
101

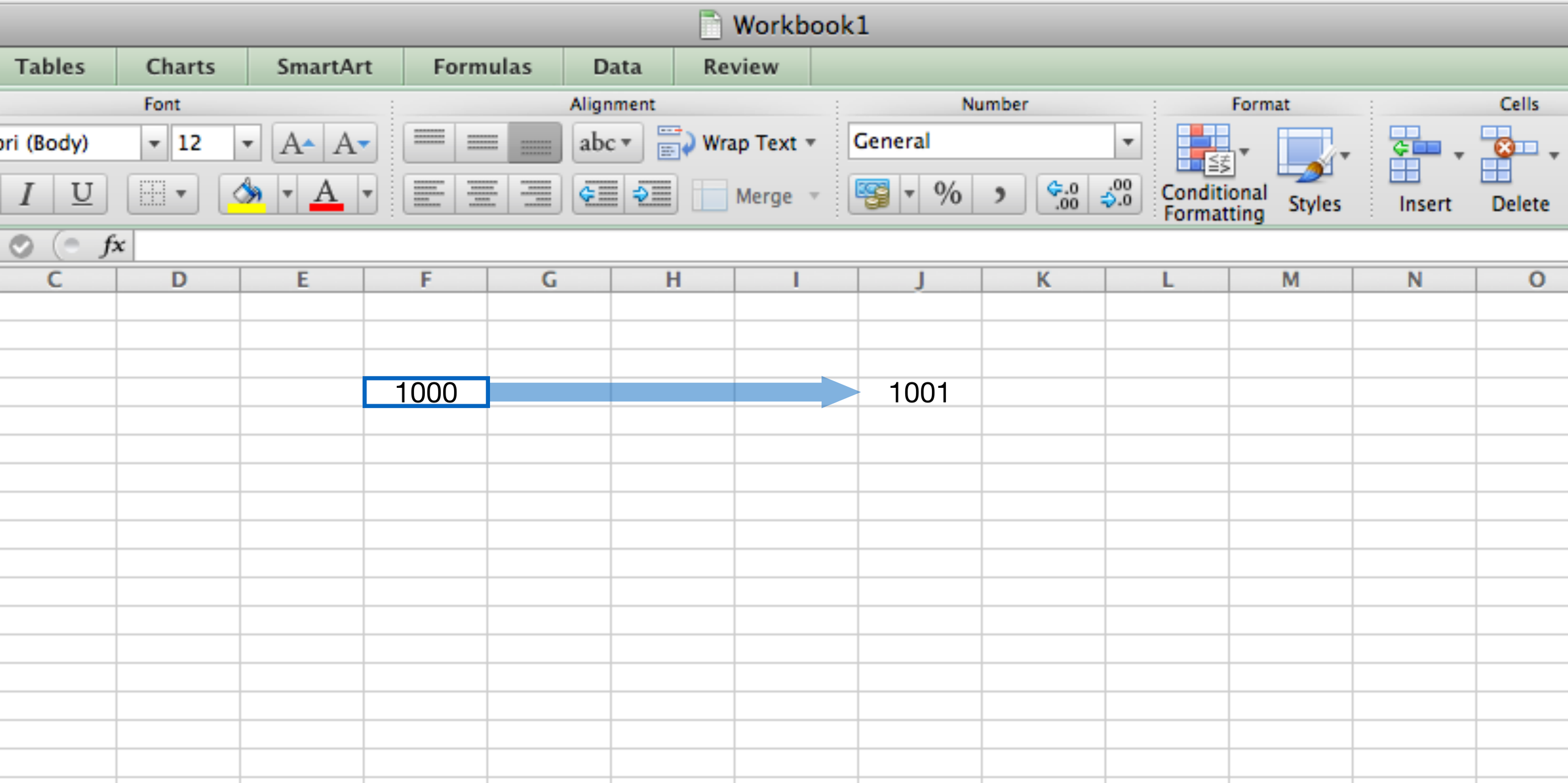

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
100
101

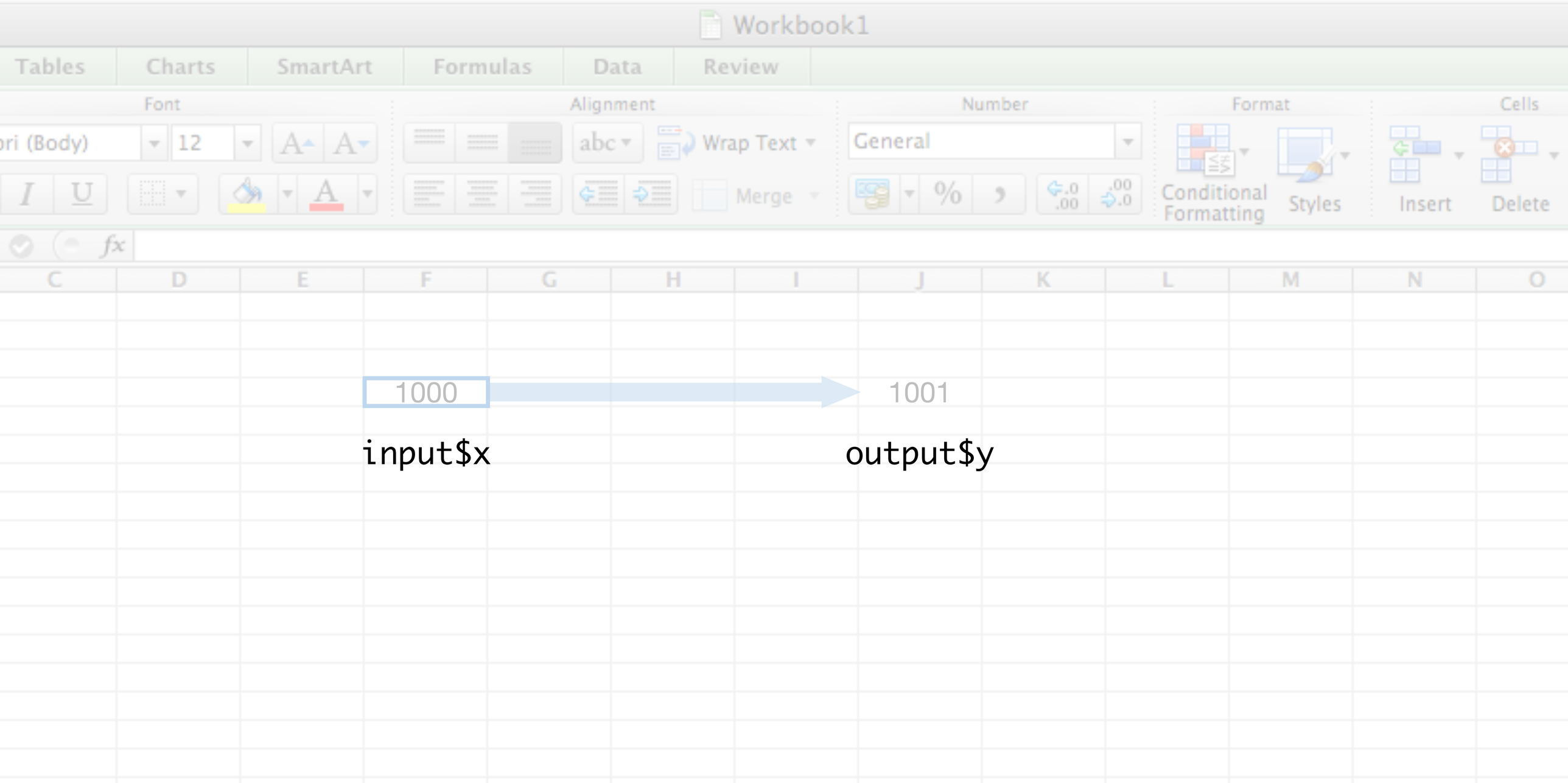

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
C
D
E
F
G
H
I
J
K
L
M
N
O
999
1000

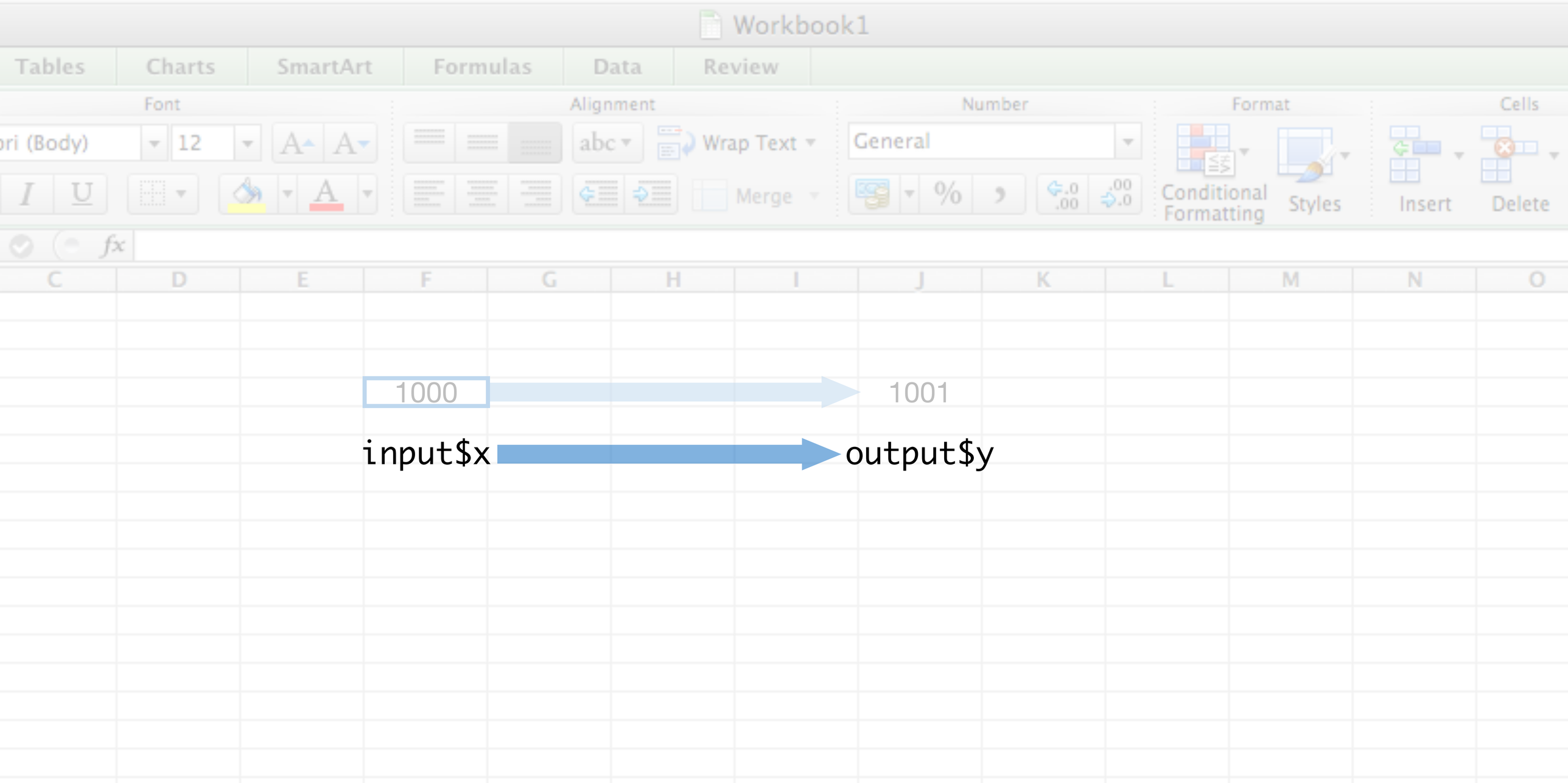

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
C
D
E
F
G
H
I
J
K
L
M
N
O
999
1000


Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
1000
1001

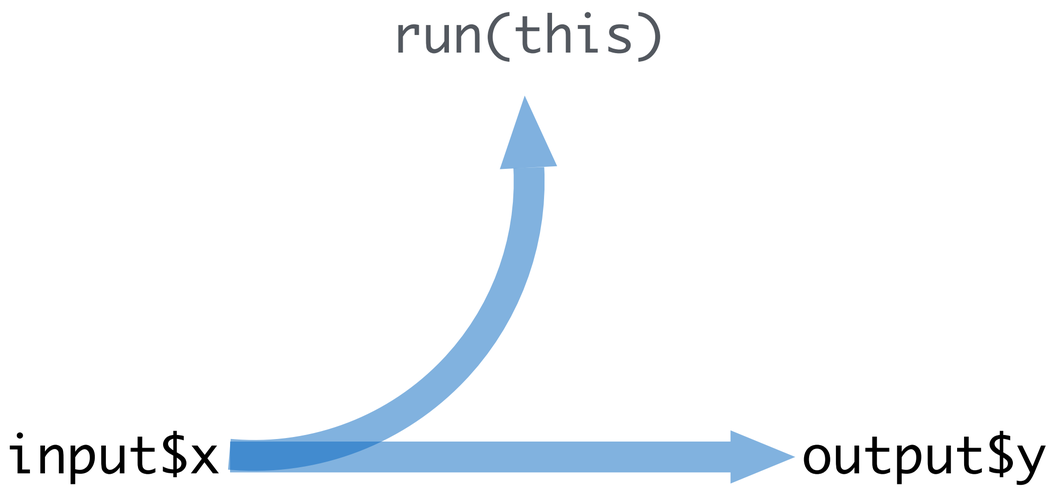

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
fx
C
D
E
F
G
H
I
J
K
L
M
N
O
1000
1001

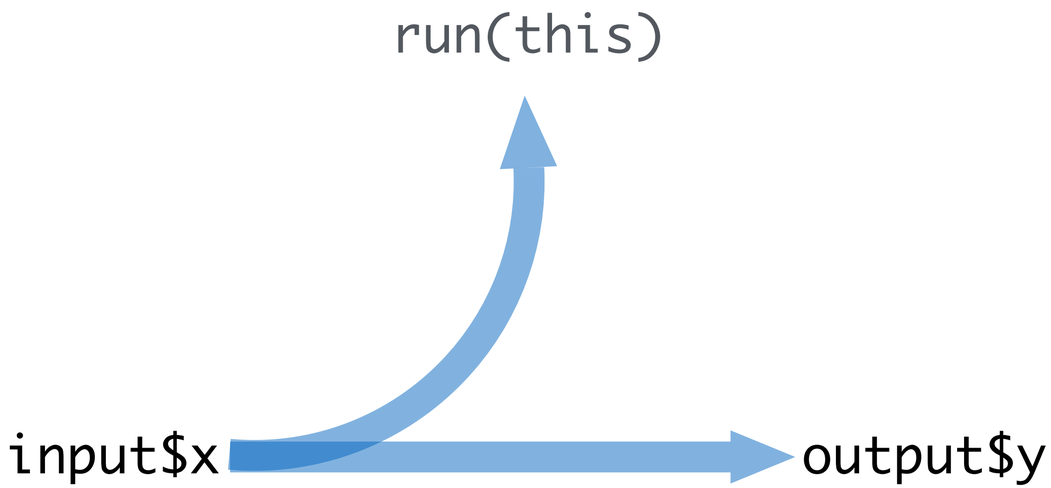

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ori (Body)
12
abc
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
fx
C
D
E
F
G
H
I
J
K
L
M
N
O
1000
1001

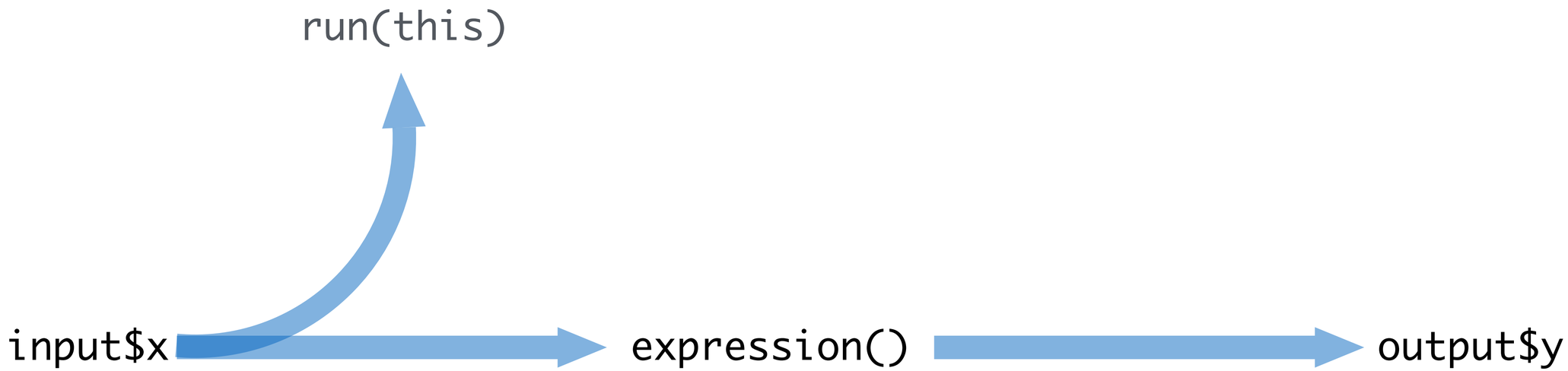

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
Wrap Text
General
Conditional Formatting
Styles
Insert
Delete
1000
1001
input$x
output$y

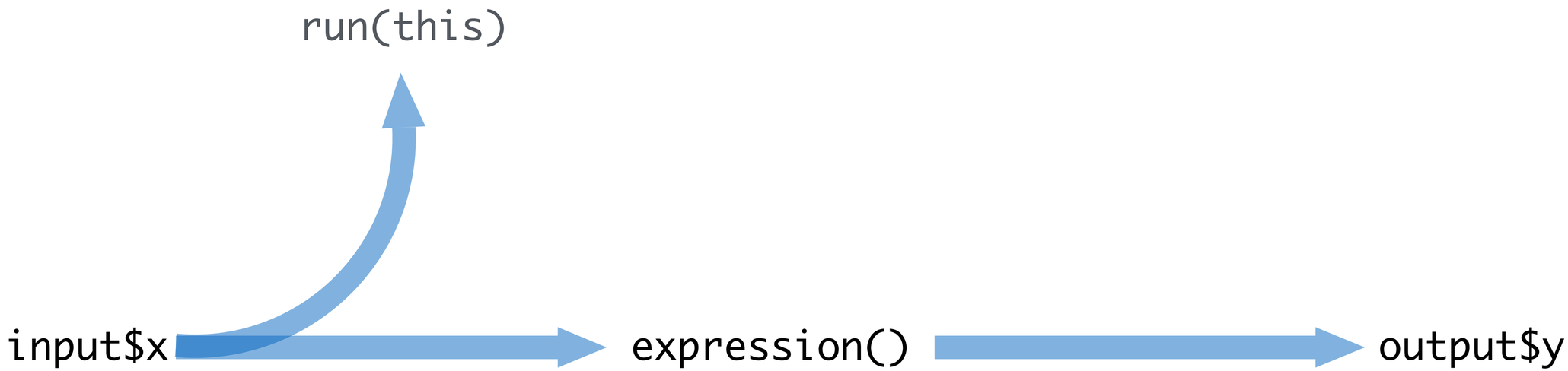

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
1000
1001
input$x
output$y

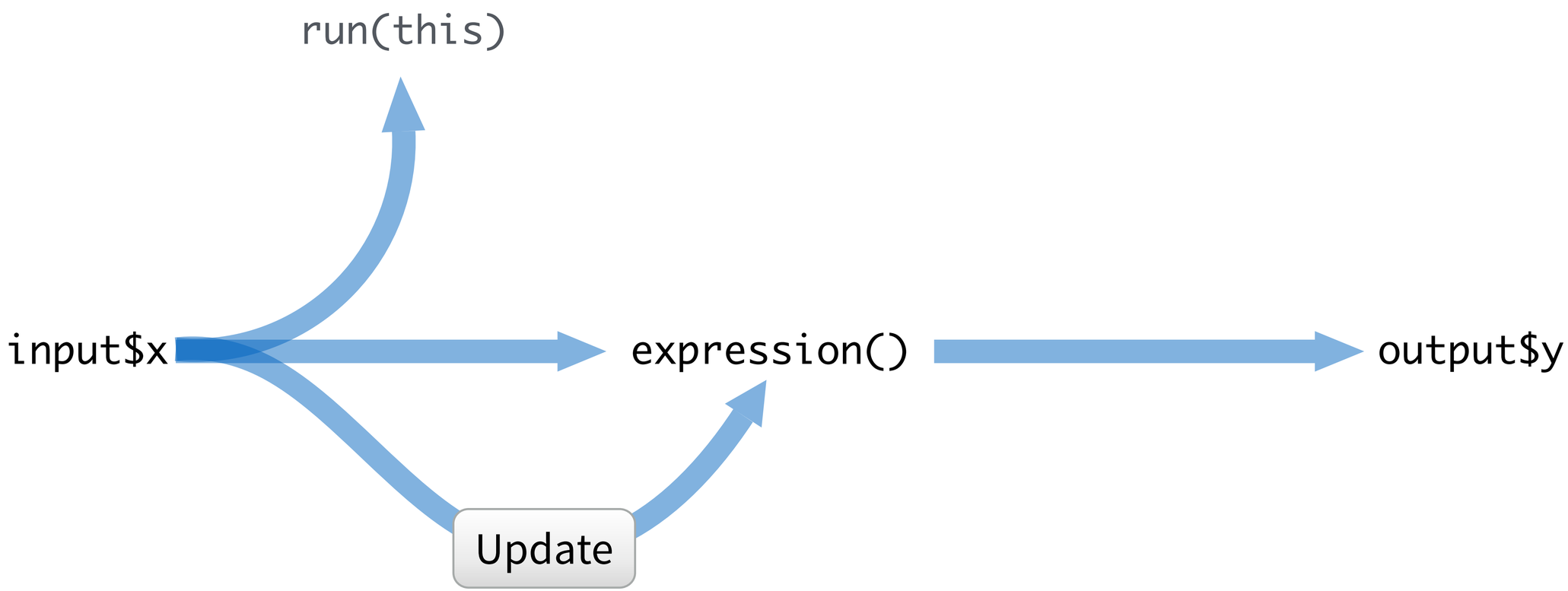

input$x
output$y

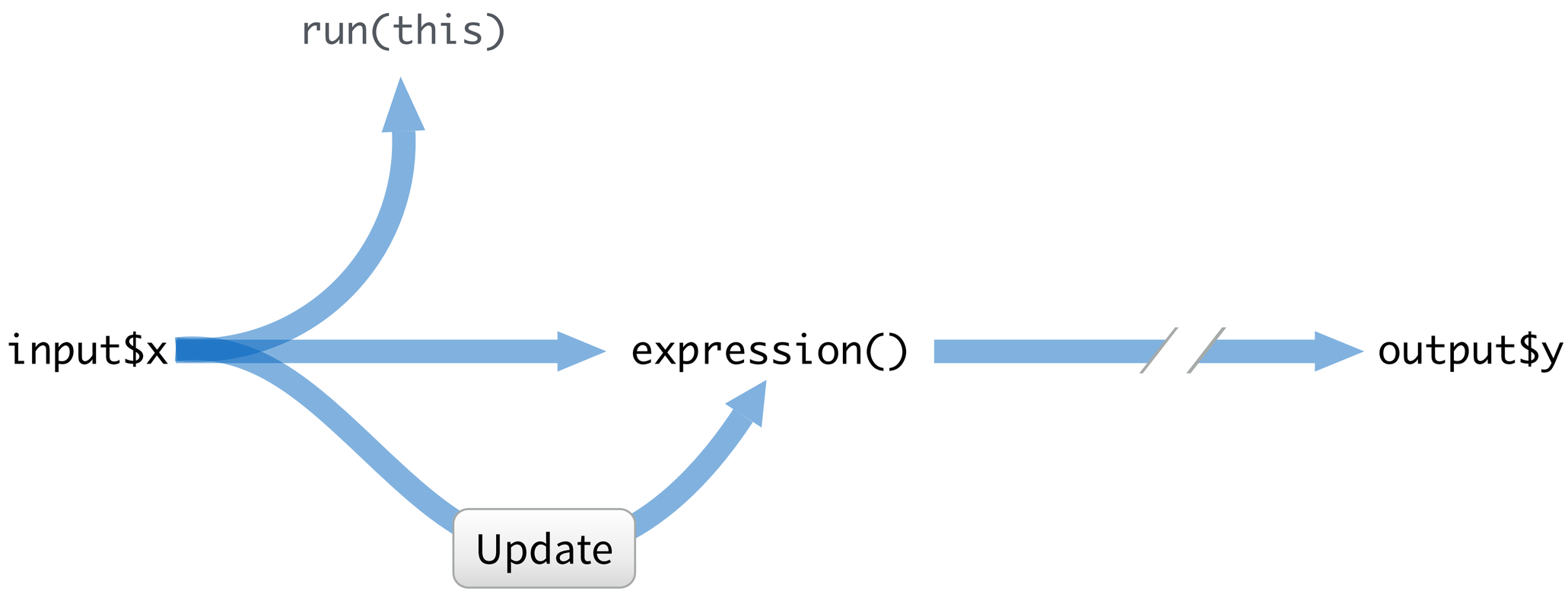

run(this)
input$x
output$y

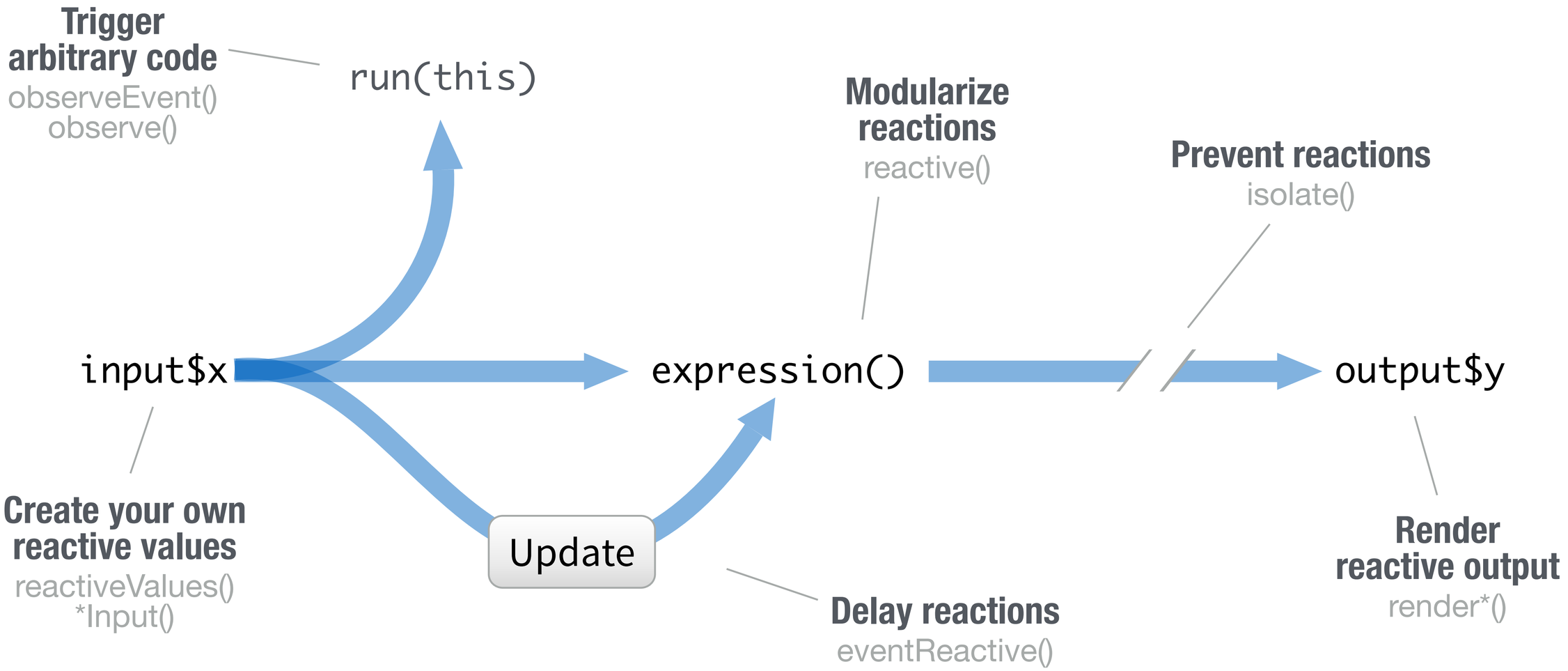

run(this)
input$x
output$y


run(this)
input$x
expression()
output$y

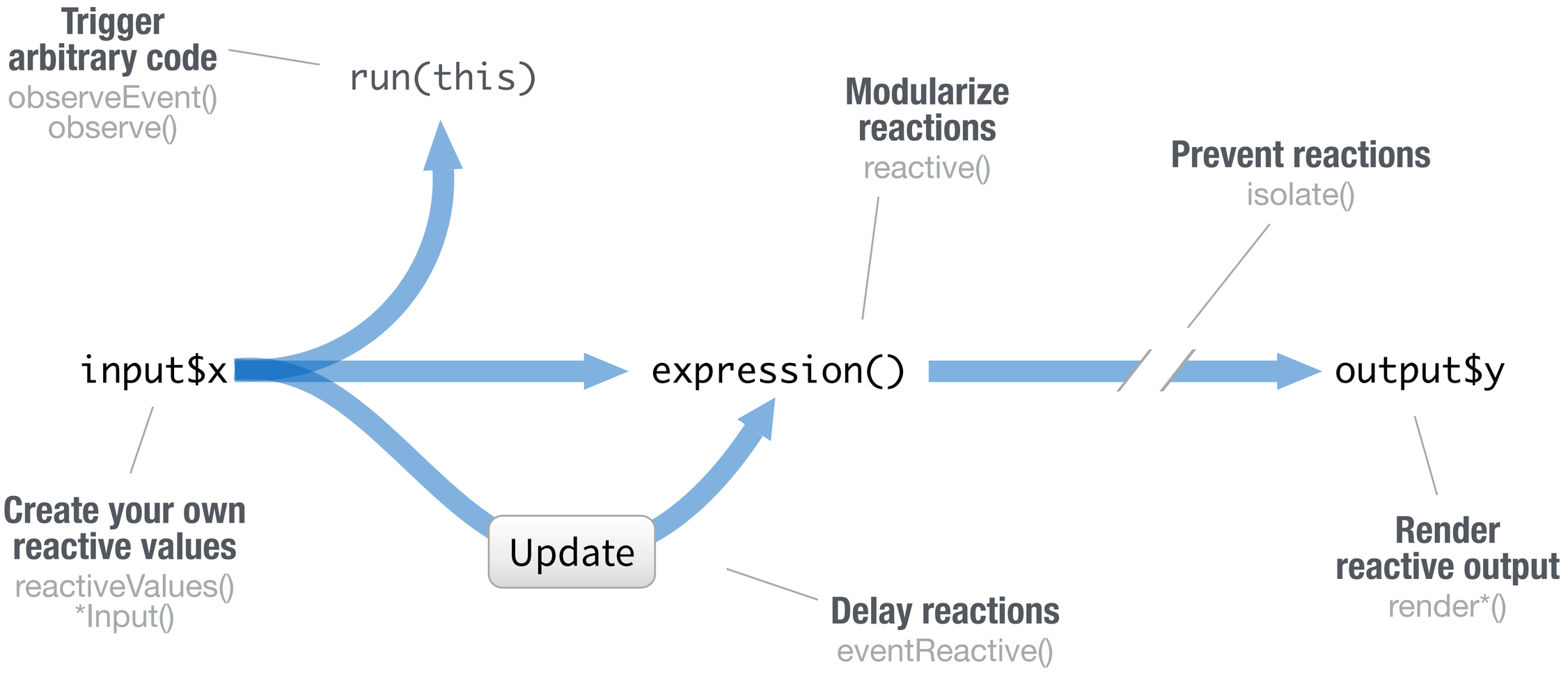

run(this)
input$x
expression()
output$y


run(this)
input$x
expression()
output$y
Update

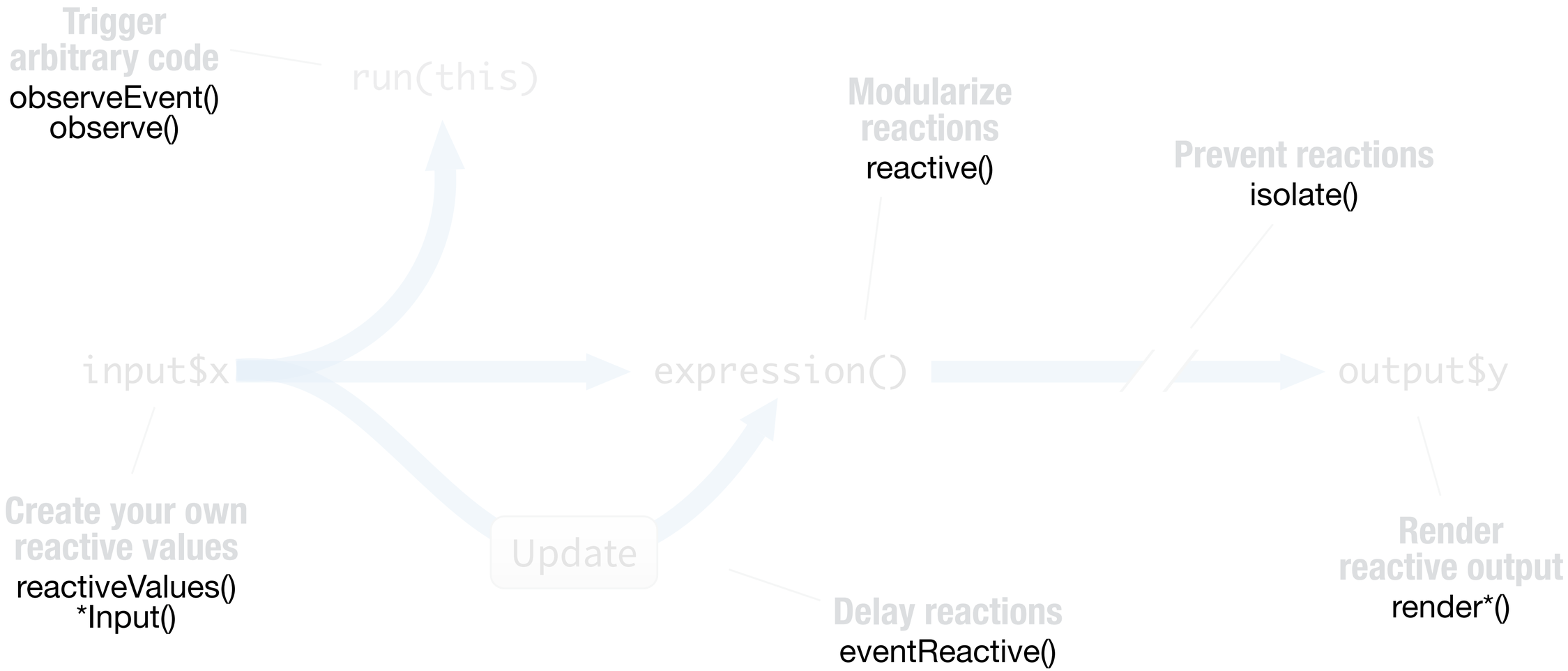

run(this)
input$x
expression()
output$y
Update


Trigger arbitrary code
observeEvent()
observe()
run(this)
Modularize reactions
reactive()
Prevent reactions
isolate()
input$x
expression()
output$y
Create your own reactive values
reactiveValues()
*Input()
Update
Delay reactions
eventReactive()
Render reactive output
render*()

Reactive functions



Reactive functions



You cannot call an **input value** (reactive value) from outside of a **reactive function**.

```
renderPlot({ hist(rnorm(input$num)) })
```

You cannot call an **input value** (reactive value) from outside of a **reactive function**.

```
renderPlot({ hist(rnorm(input$num)) })
```

```
hist(rnorm(input$num))
```

Think of reactivity in R as a two step process

`input$x` `output$y`

Think of reactivity in R as a two step process

1 Reactive values notify
the objects that use them
when they become invalid

`input$x` `output$y`

Think of reactivity in R as a two step process

1 Reactive values notify
the objects that use them
when they become invalid

Think of reactivity in R as a two step process

1 Reactive values notify
the objects that use them
when they become invalid

2 Objects respond
How the object responds
depends on which reactive
function created it.

Think of reactivity in R as a two step process

1 Reactive values notify
the objects that use them when they become invalid

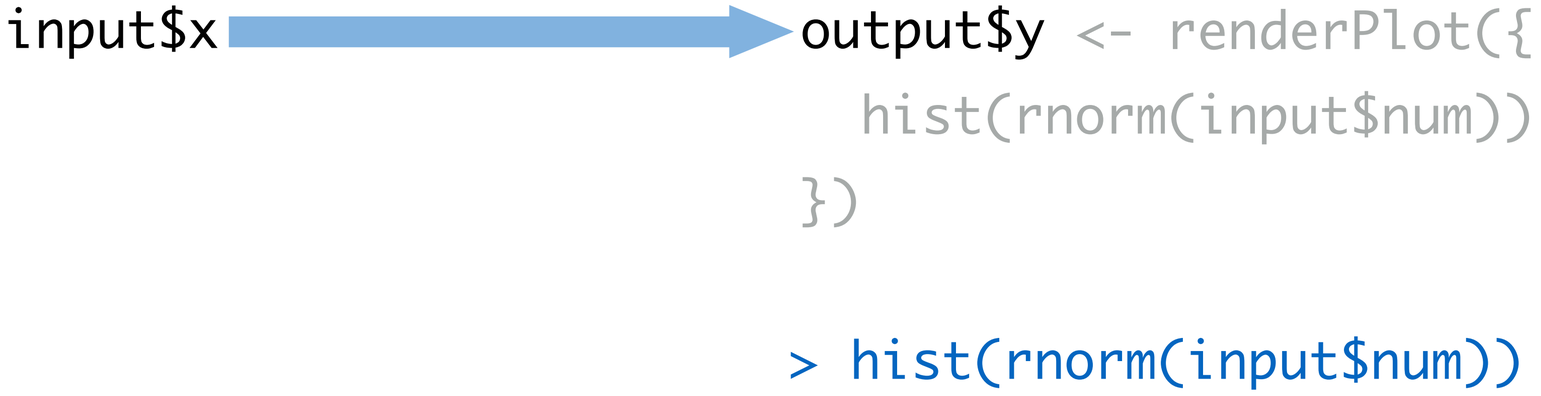
2 Objects respond
How the object responds depends on which reactive function created it.


```
output$y <- renderPlot({
  hist(rnorm(input$num))
})
```

Think of reactivity in R as a two step process

1 Reactive values notify
the objects that use them when they become invalid

2 Objects respond
How the object responds depends on which reactive function created it.



render*()

```
output$p <- renderPlot({hist(rnorm(input$num))})
```

Builds an object that:

When notified by:

render*()

```
output$p <- renderPlot({hist(rnorm(input$num))})
```

Builds an object that:

Reruns code chunk
(saves results to output$)

When notified by:

render*()

```
output$p <- renderPlot({hist(rnorm(input$num))})
```

Builds an object that:

Reruns code chunk
(saves results to output$)

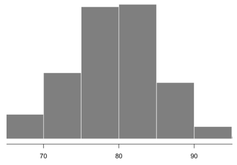
When notified by:

any reactive value in the code chunk

Each function builds a different type of output.

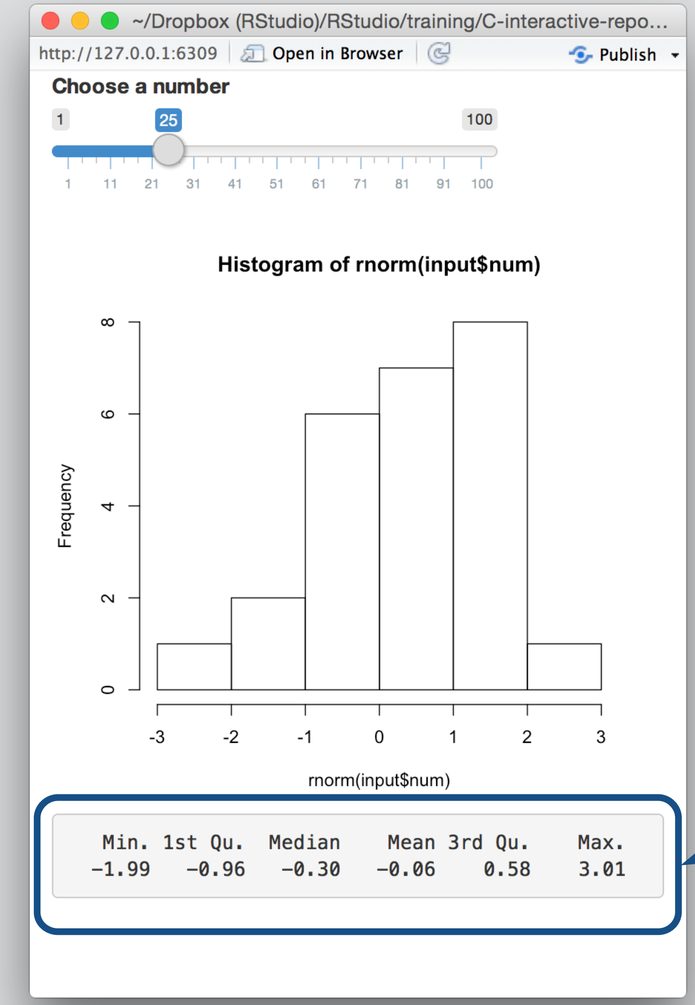
function	creates
`renderDataTable()`	An interactive table (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed R output
`renderTable()`	A table (from a data frame, matrix, or other table-like structure)
`renderText()`	A character string
`renderUI()`	a Shiny UI element

Use…



render*() to make an **object to display** in the UI.

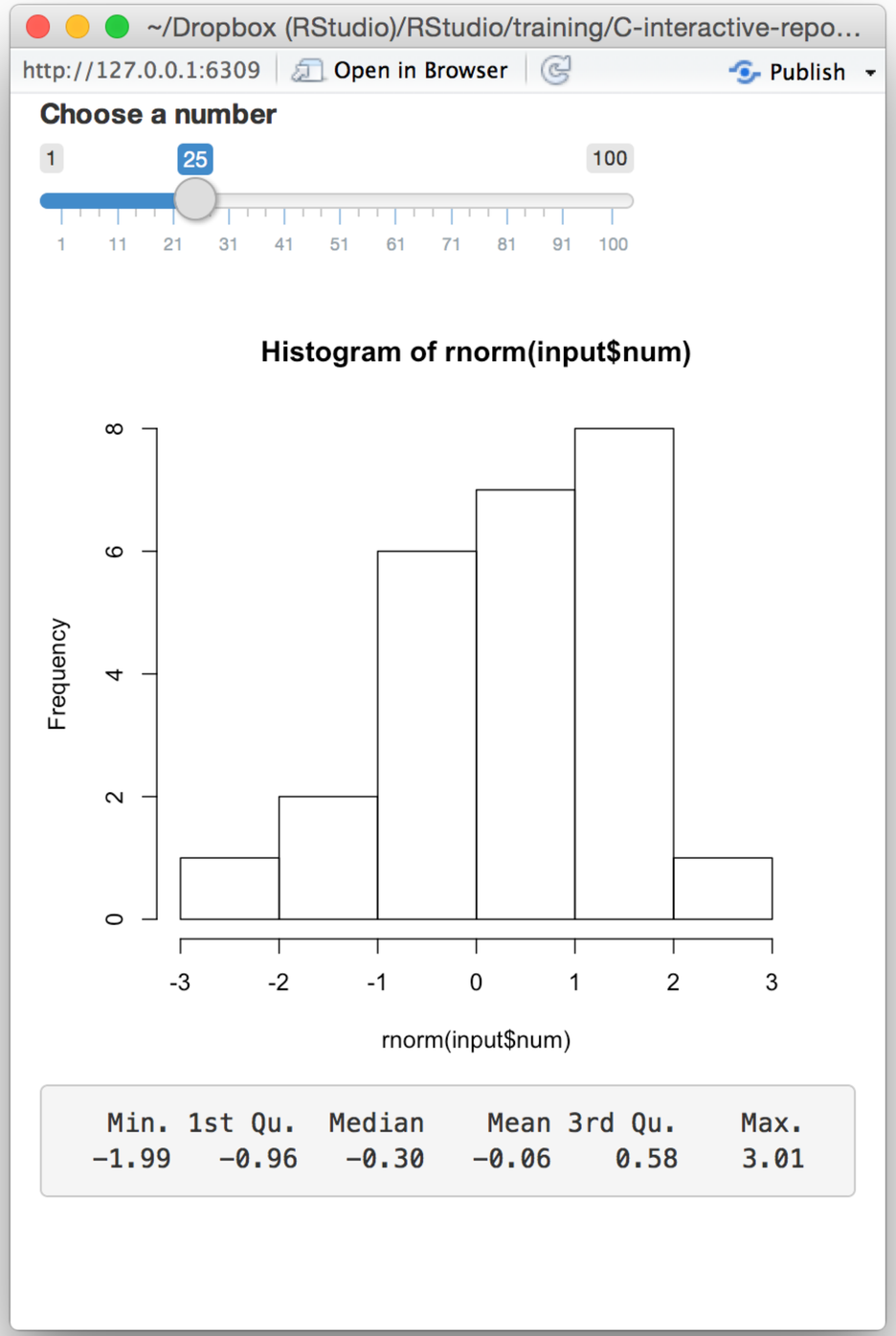
Your Turn



Use **renderPrint()** and **verbatimTextOutput()** to add a **summary()** of **rnorm(input$num)** to your app, e.g.

```
summary(rnorm(input$num))
```

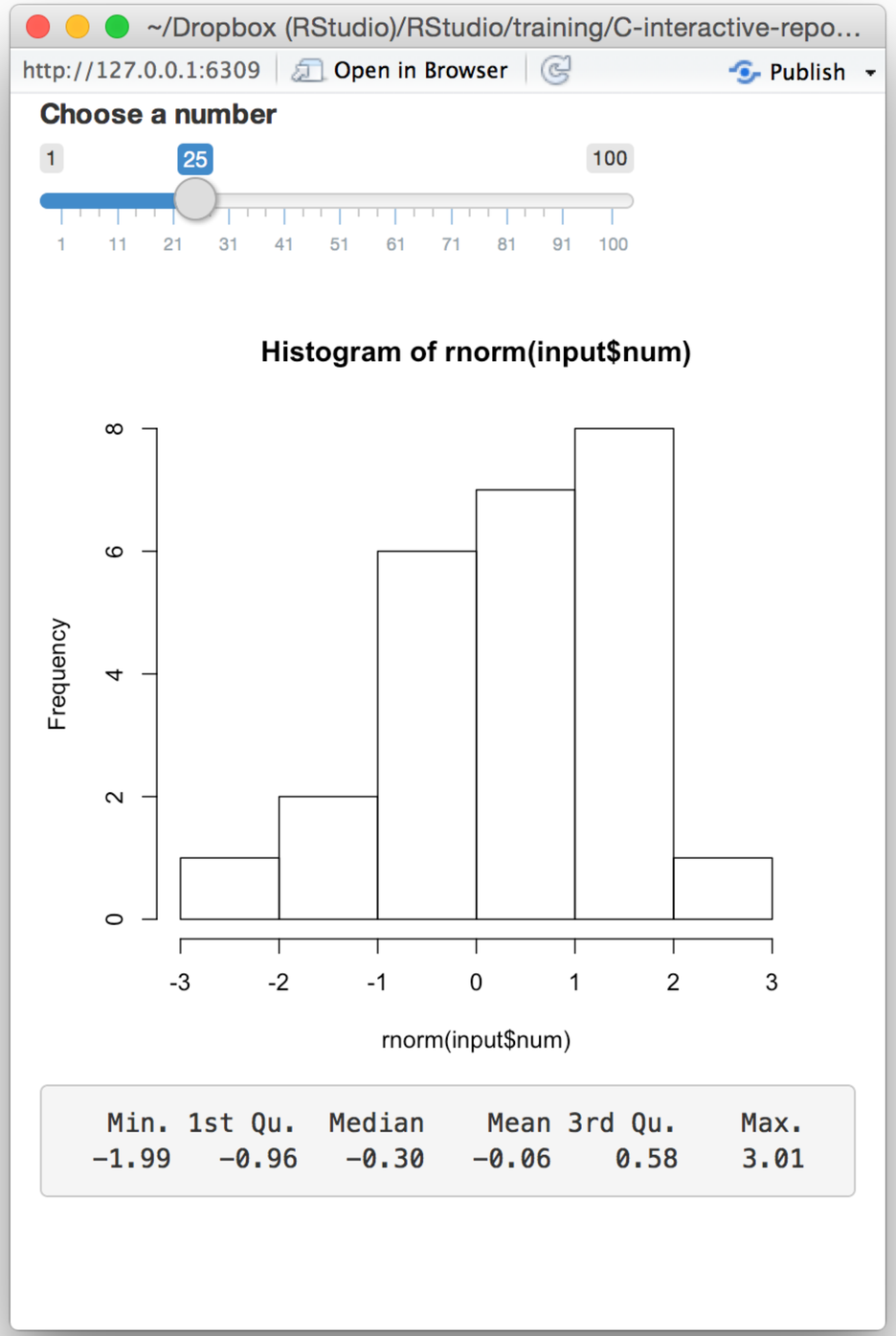
```
ui <- fluidPage(
  sliderInput("num", “Choose a #”, 1, 100, 50),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   output$hist <- renderPlot({
     hist(rnorm(input$num))
   })
   output$sum <- renderPrint({
     summary(rnorm(input$num))
   })
}
shinyApp(ui = ui, server = server)
```



input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

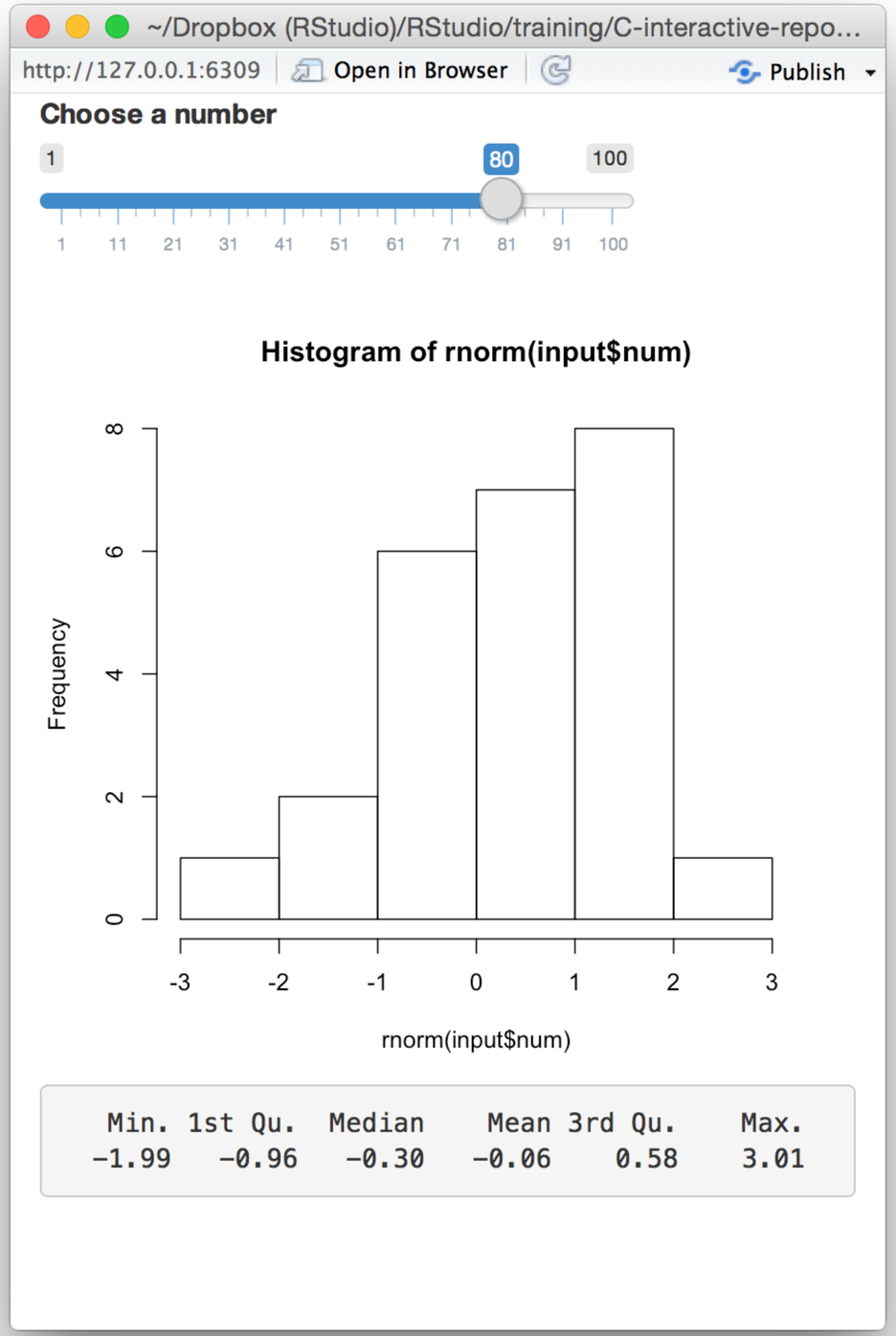
```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```



input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

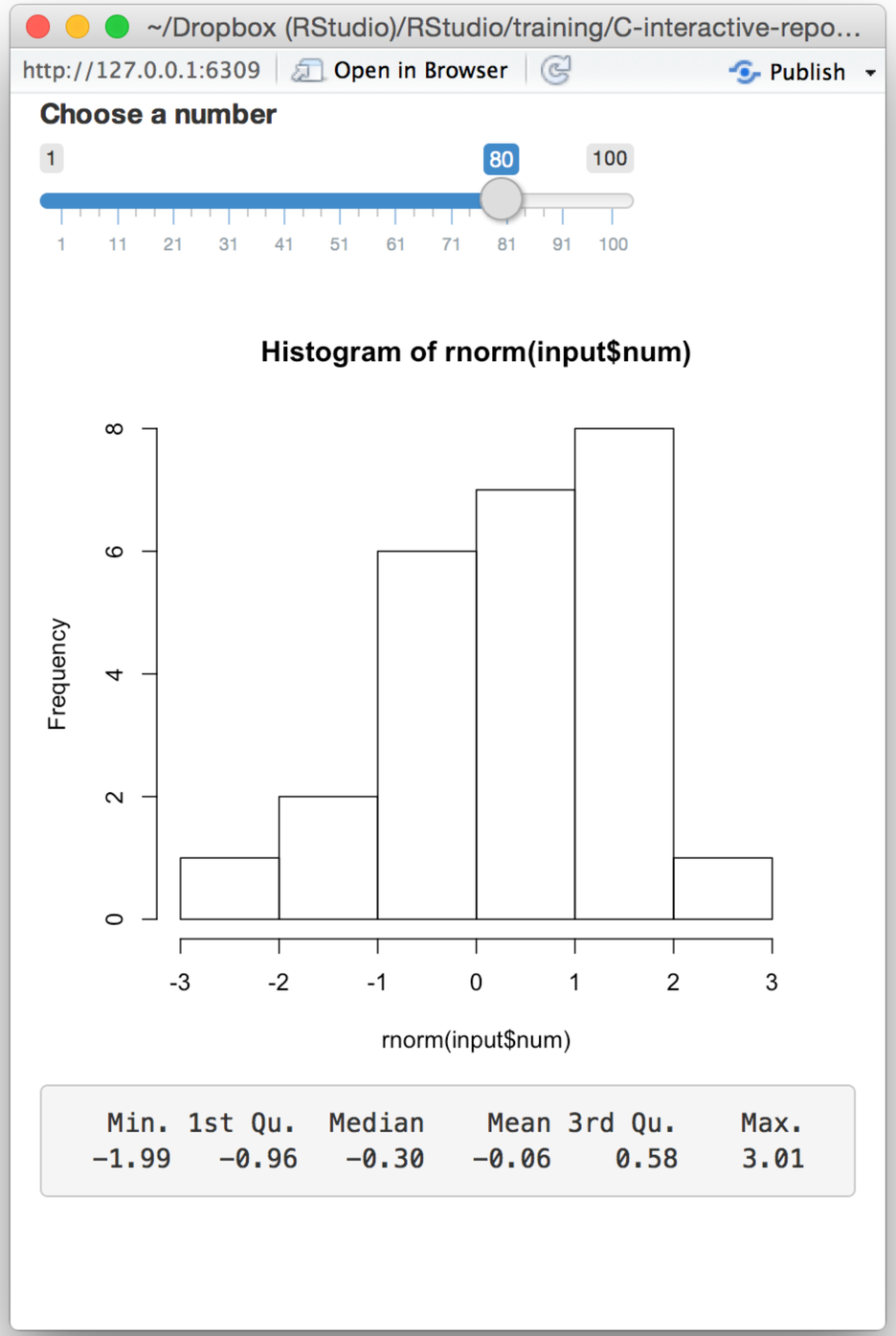
```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```



input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

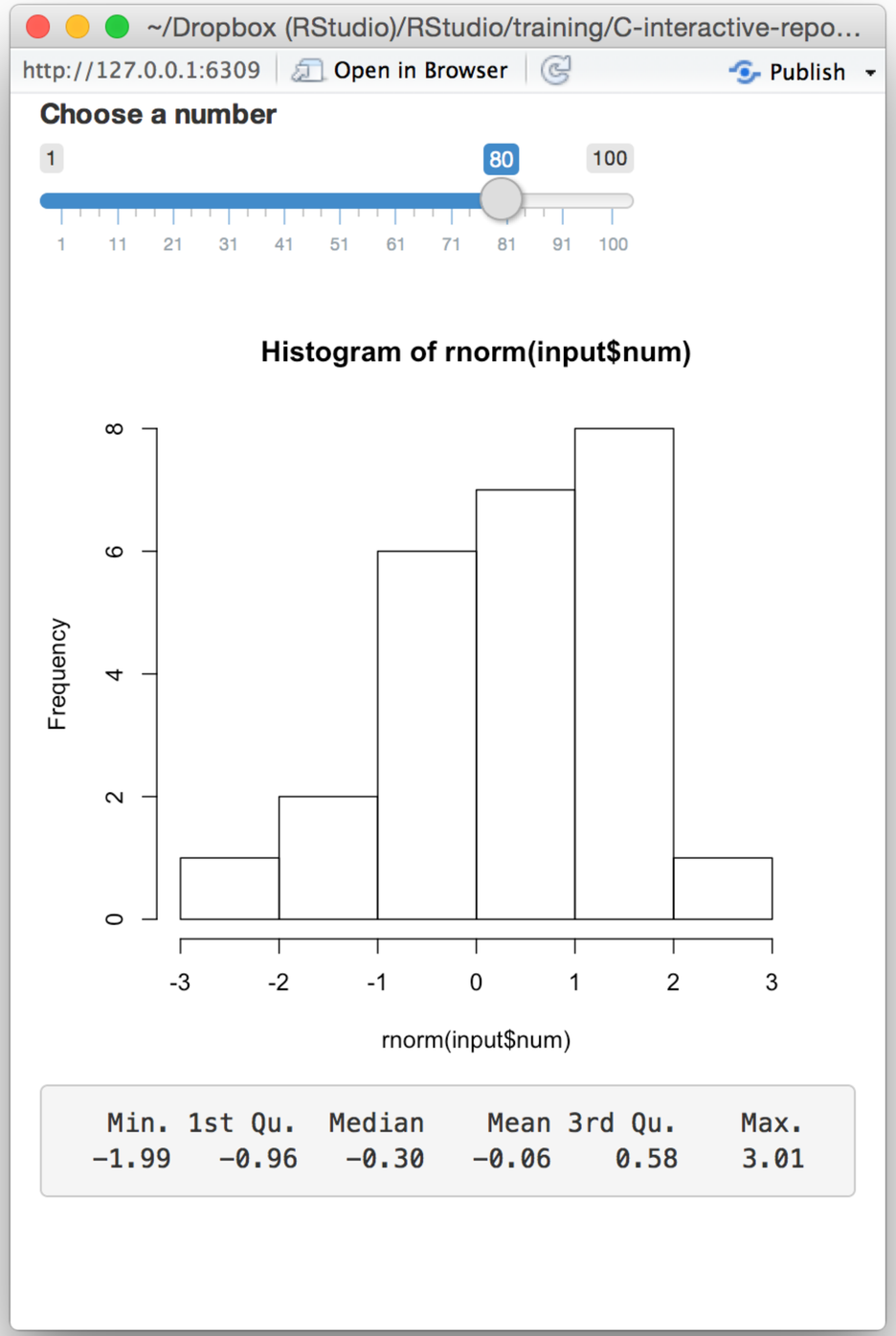
```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```



input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```

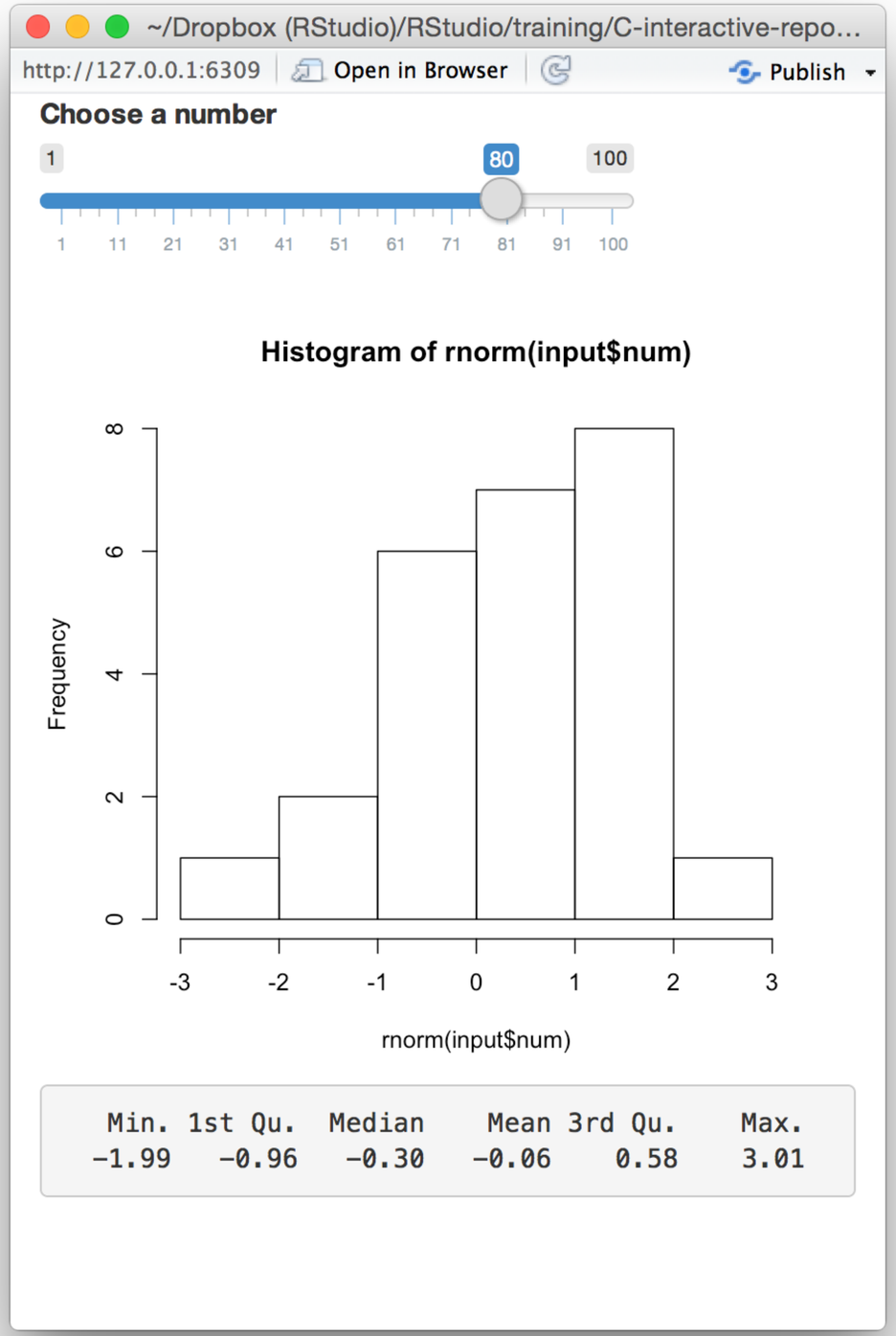


input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
    summary(rnorm(input$num))
  })
```

```
> hist(rnorm(input$num))
```

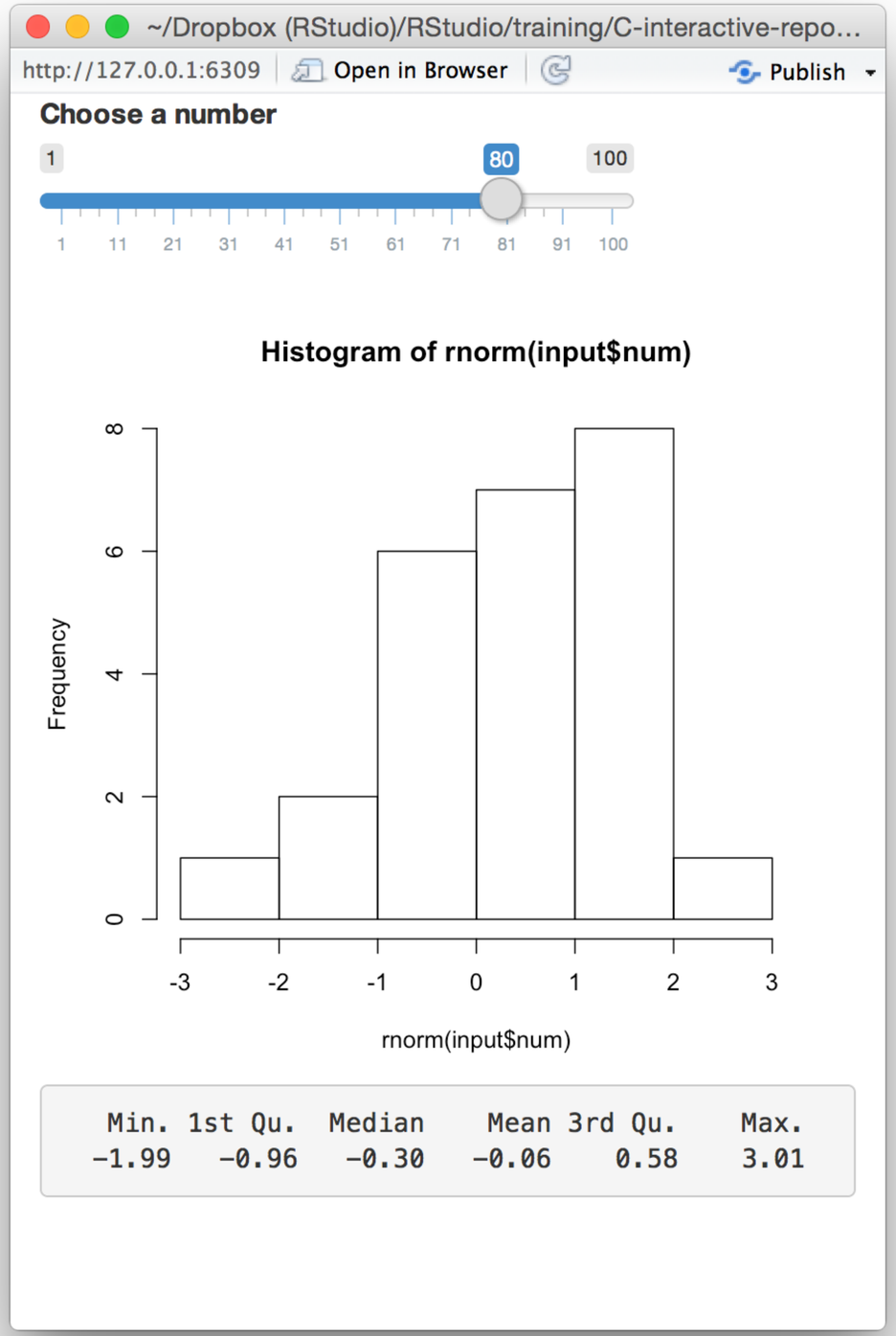


input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
    summary(rnorm(input$num))
  })
```

```
> hist(rnorm(input$num))
```

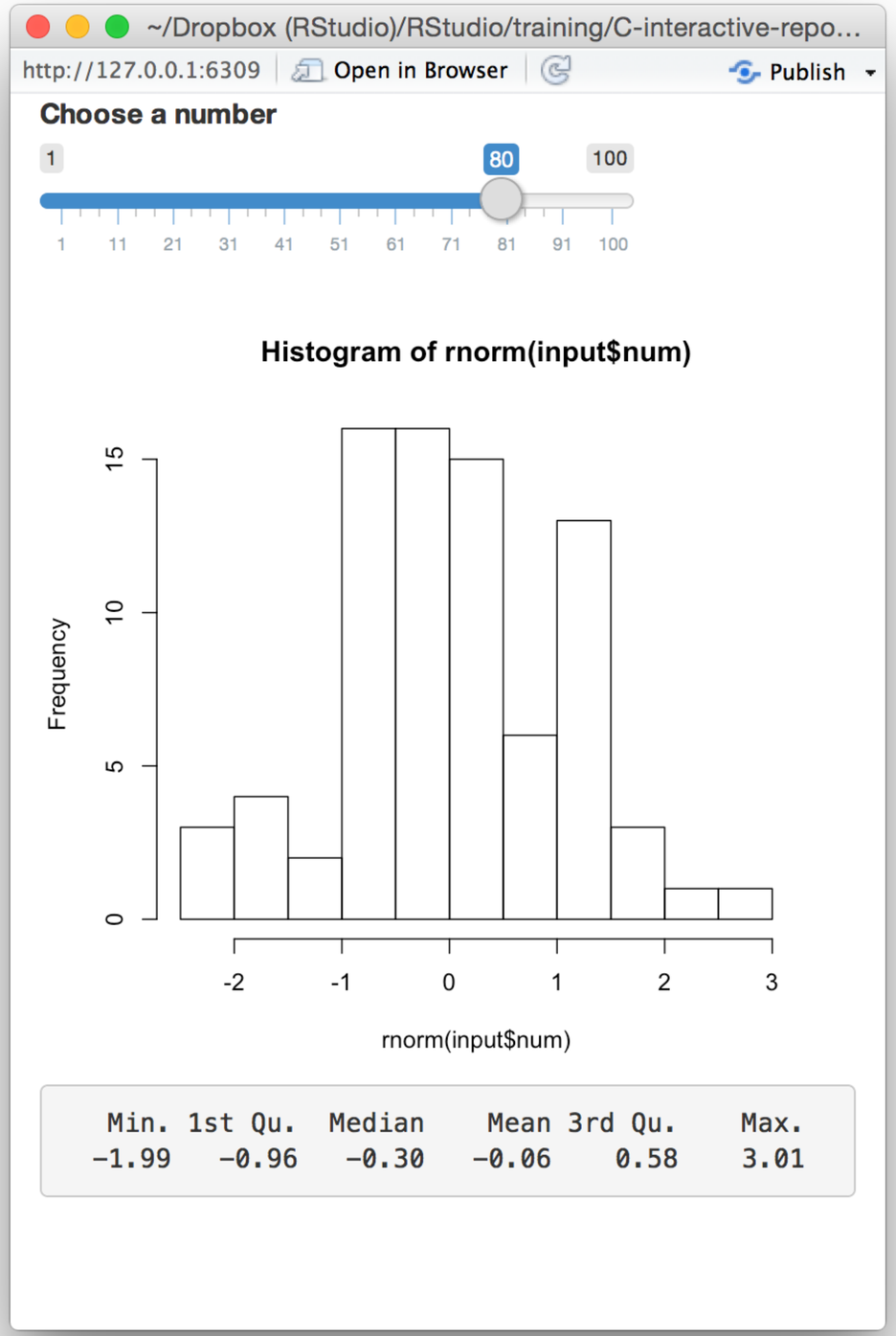


input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```

```
> hist(rnorm(input$num))
```



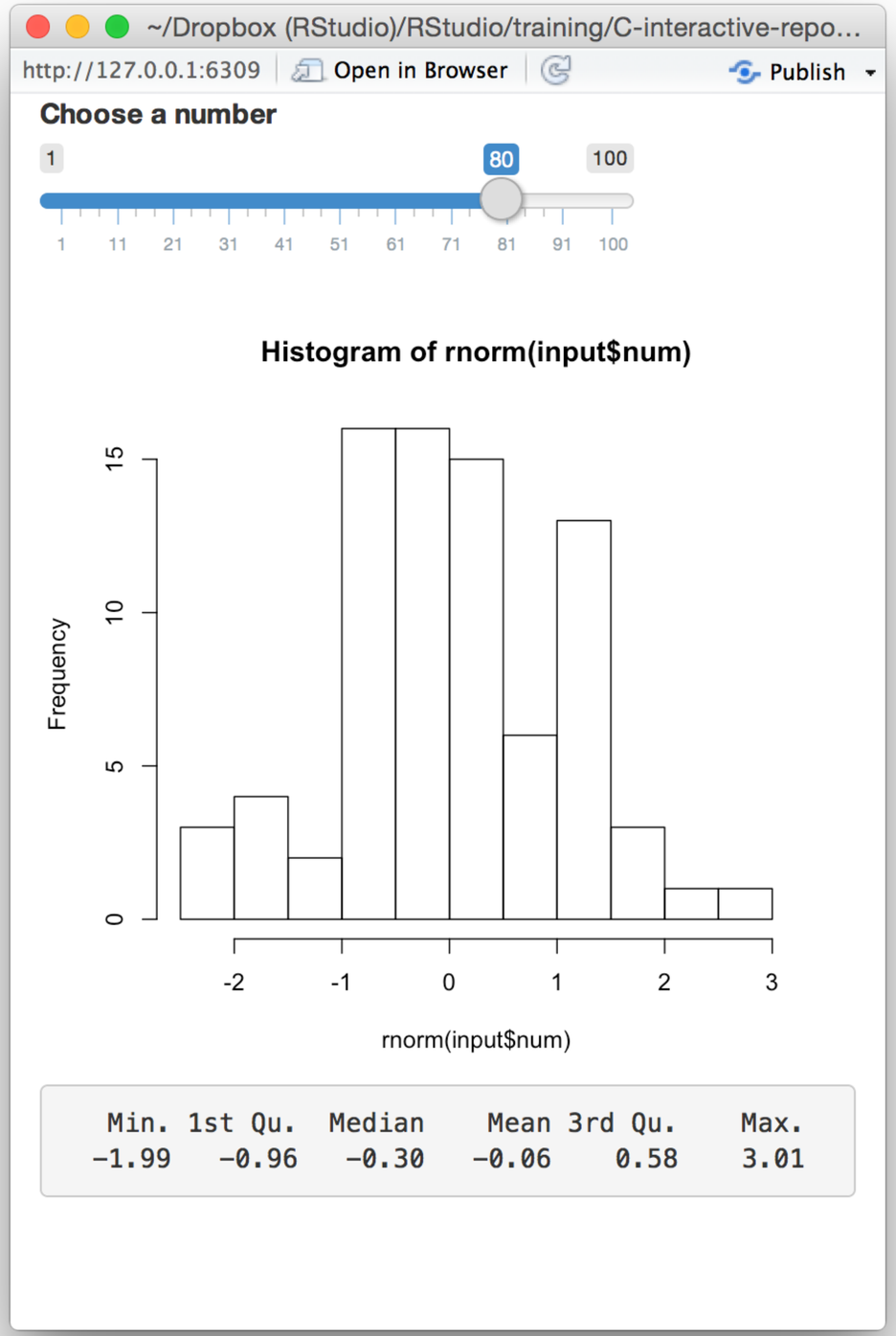
input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```

```
> hist(rnorm(input$num))
```

```
> summary(rnorm(input$num))
```



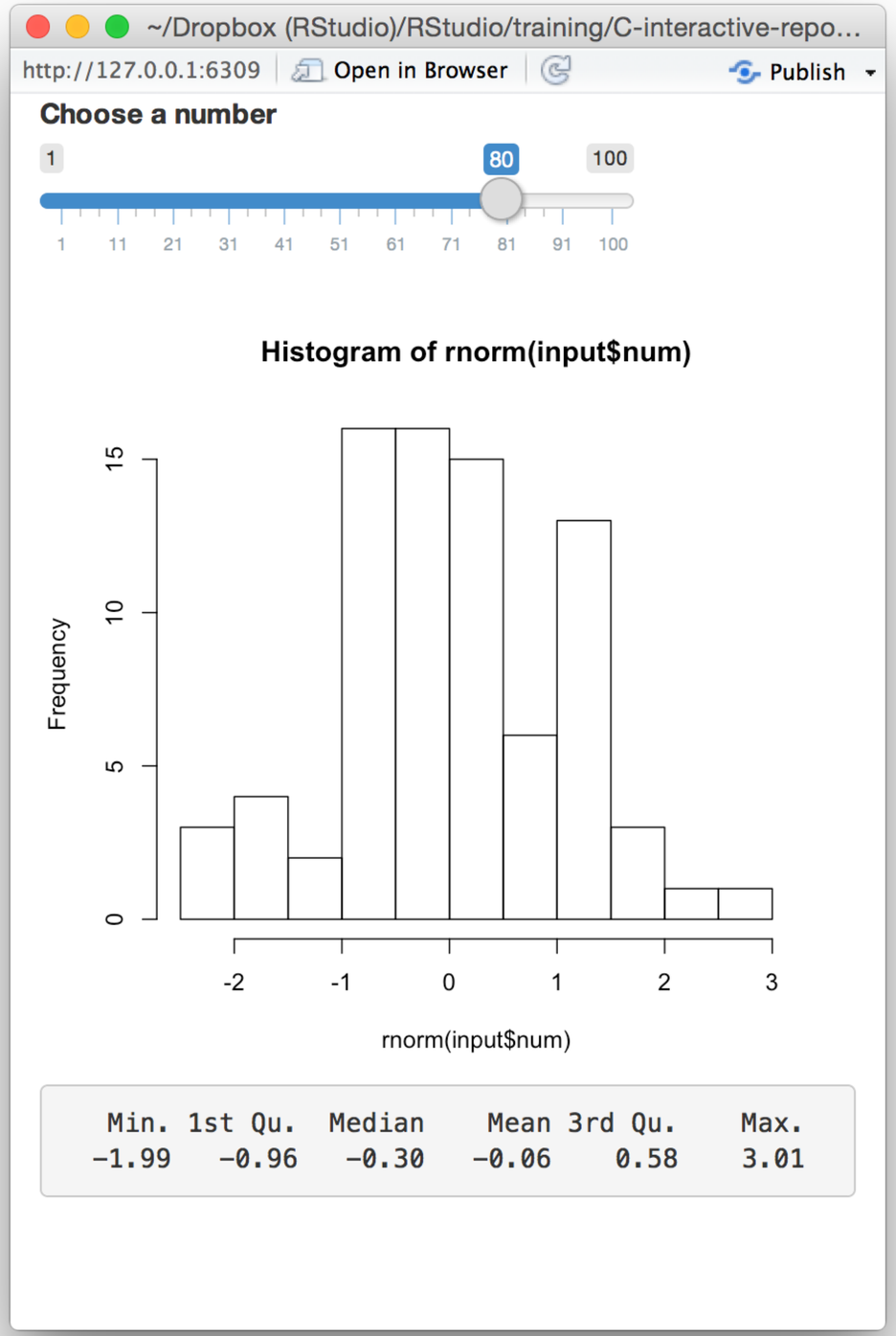
input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
    summary(rnorm(input$num))
  })
```

```
> hist(rnorm(input$num))
```

```
> summary(rnorm(input$num))
```



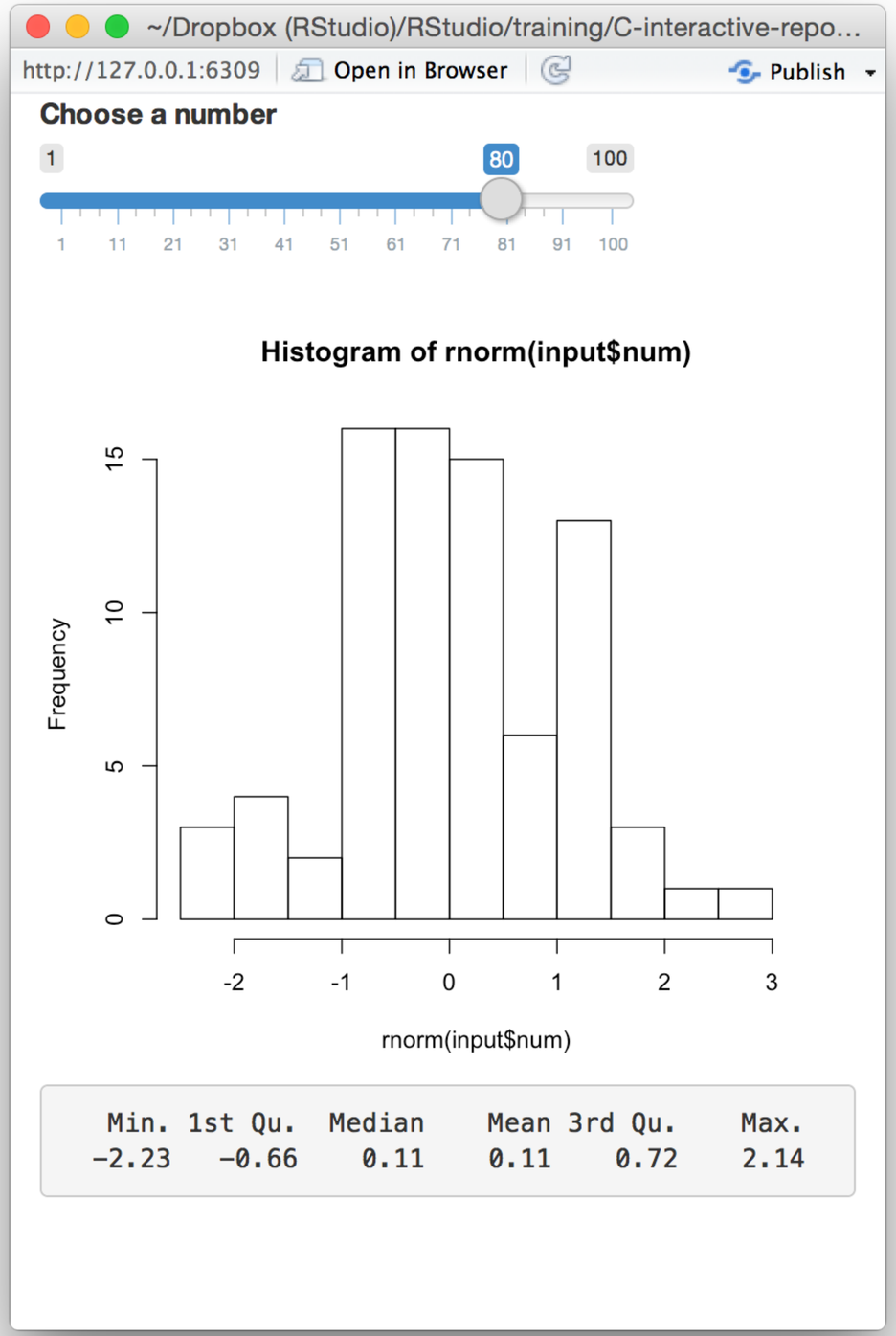
input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
    summary(rnorm(input$num))
  })
```

```
> hist(rnorm(input$num))
```

```
> summary(rnorm(input$num))
```



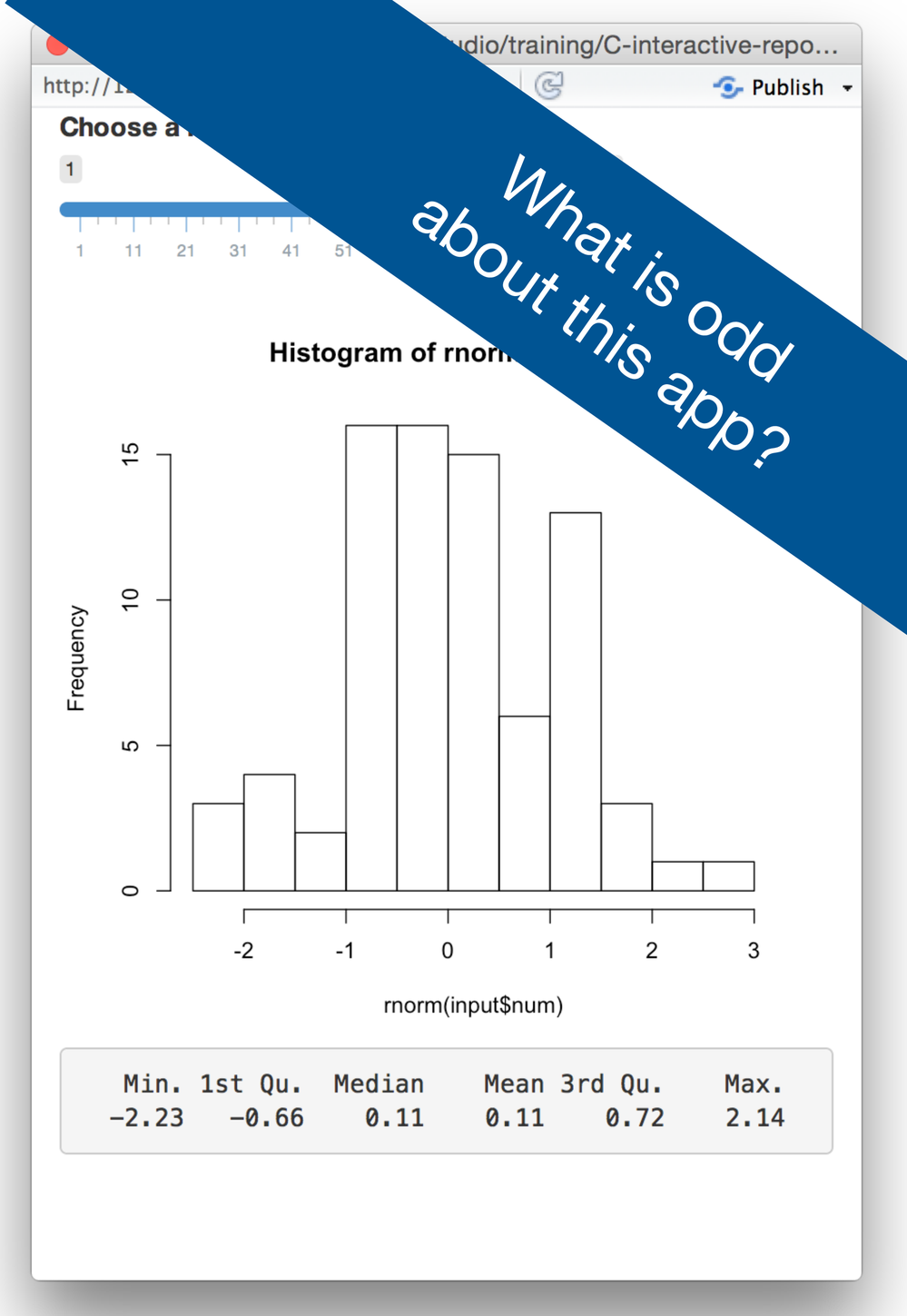
input$num

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```

```
> hist(rnorm(input$num))
```

```
> summary(rnorm(input$num))
```

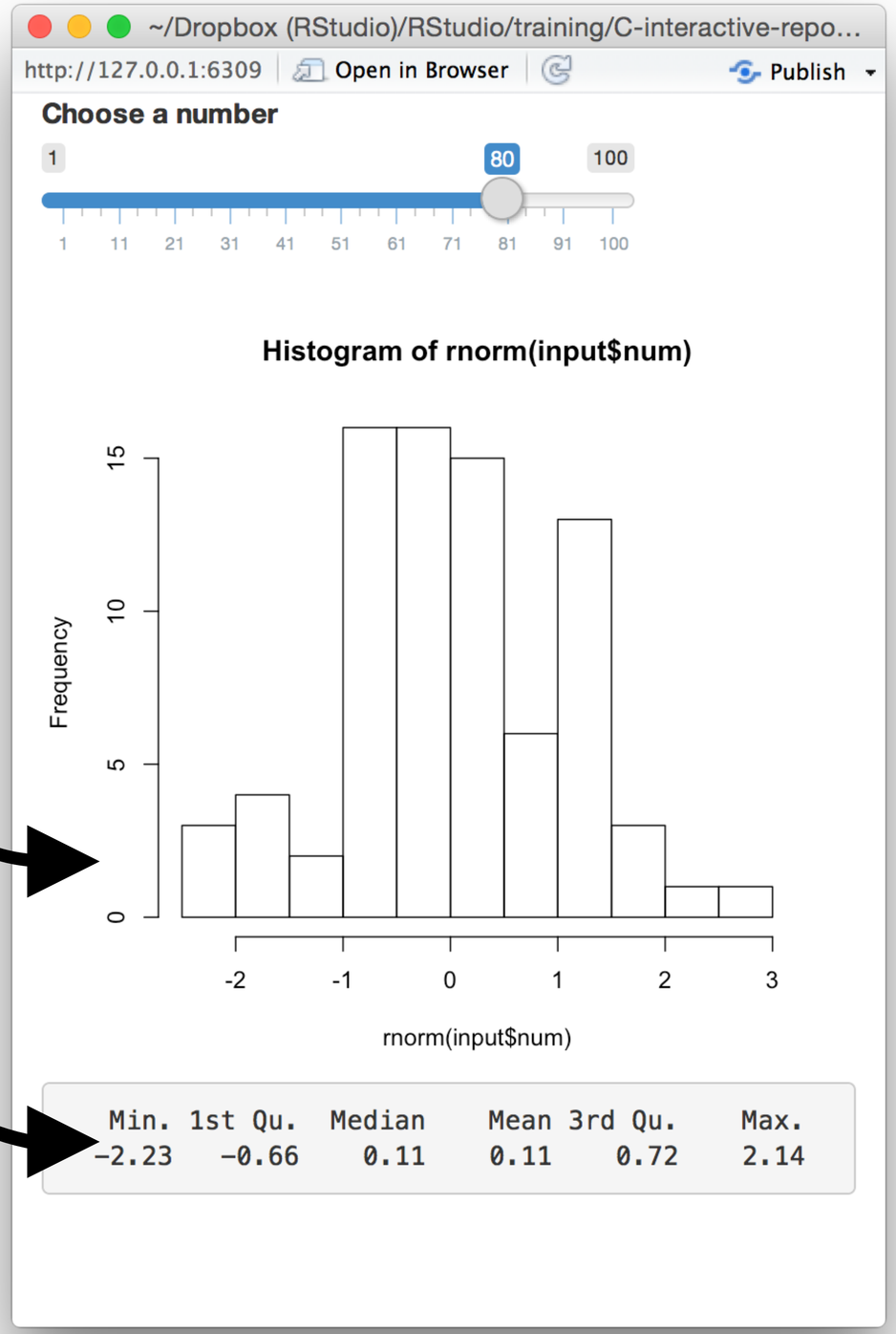


input$num

Do these describe the same data?

```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

```
output$stats <-
  renderPrint({
    summary(rnorm(input$num))
  })
```



reactive()

Makes a reactive object that you can use in downstream code.

```
data <- reactive(   { rnorm(input$num) })
```

reactive()

Makes a reactive object that you can use in downstream code.

```
data <- reactive(   { rnorm(input$num) })
```

Builds an object that:

notifies objects that use it that they are invalid

reactive()

Makes a reactive object that you can use in downstream code.

```
data <- reactive(   { rnorm(input$num) })
```

Builds an object that: notifies objects that use it that they are invalid

When notified by: any reactive value in the code chunk

A reactive expression is special in two ways

```
data()
```

1. You call a reactive expression like a function

A reactive expression is special in two ways

```
data()
```

1. You call a reactive expression like a function

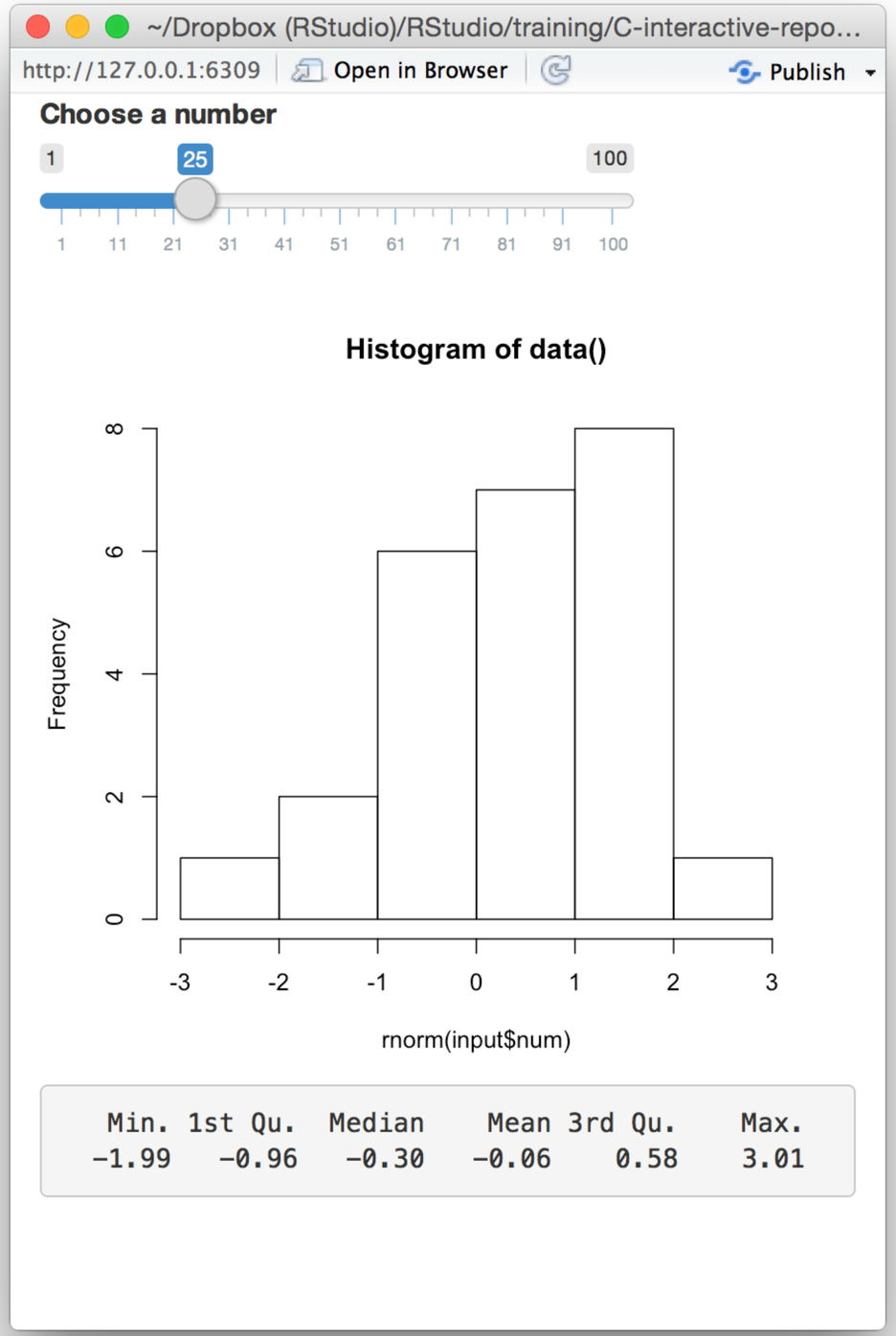
2. Reactive expressions **cache** their values
(the expression will return its most recent value, unless it has become invalidated)

```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

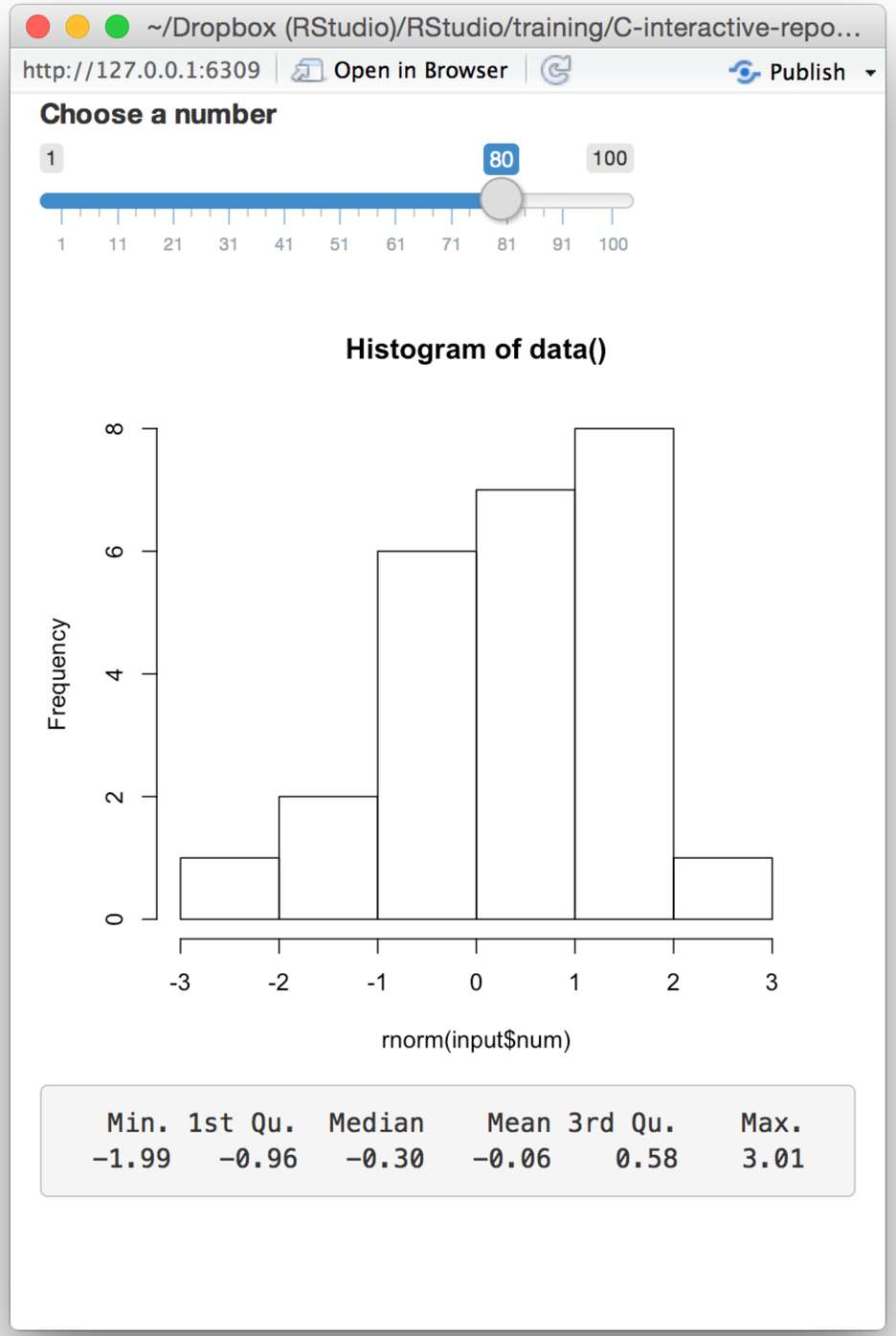


```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

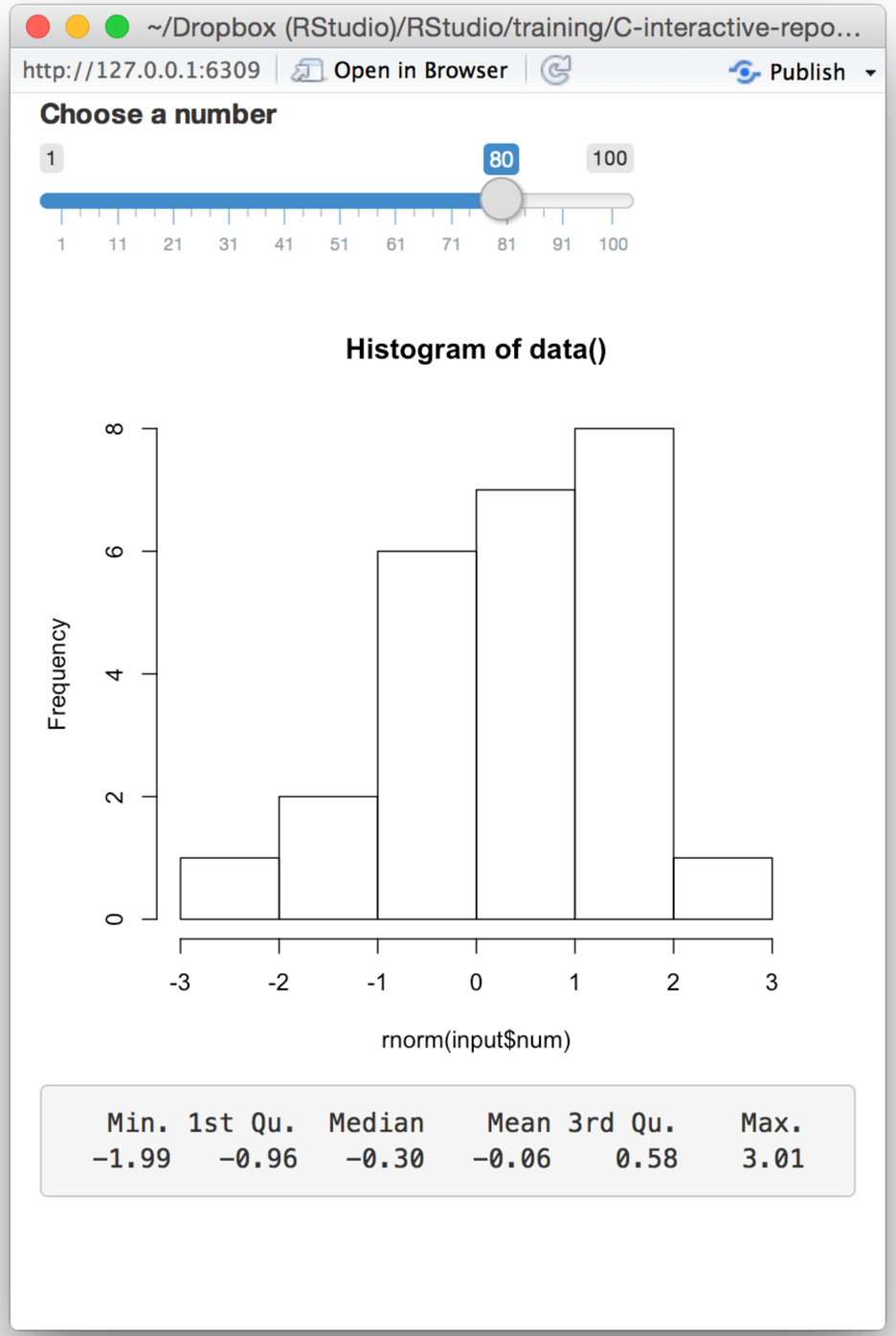


```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

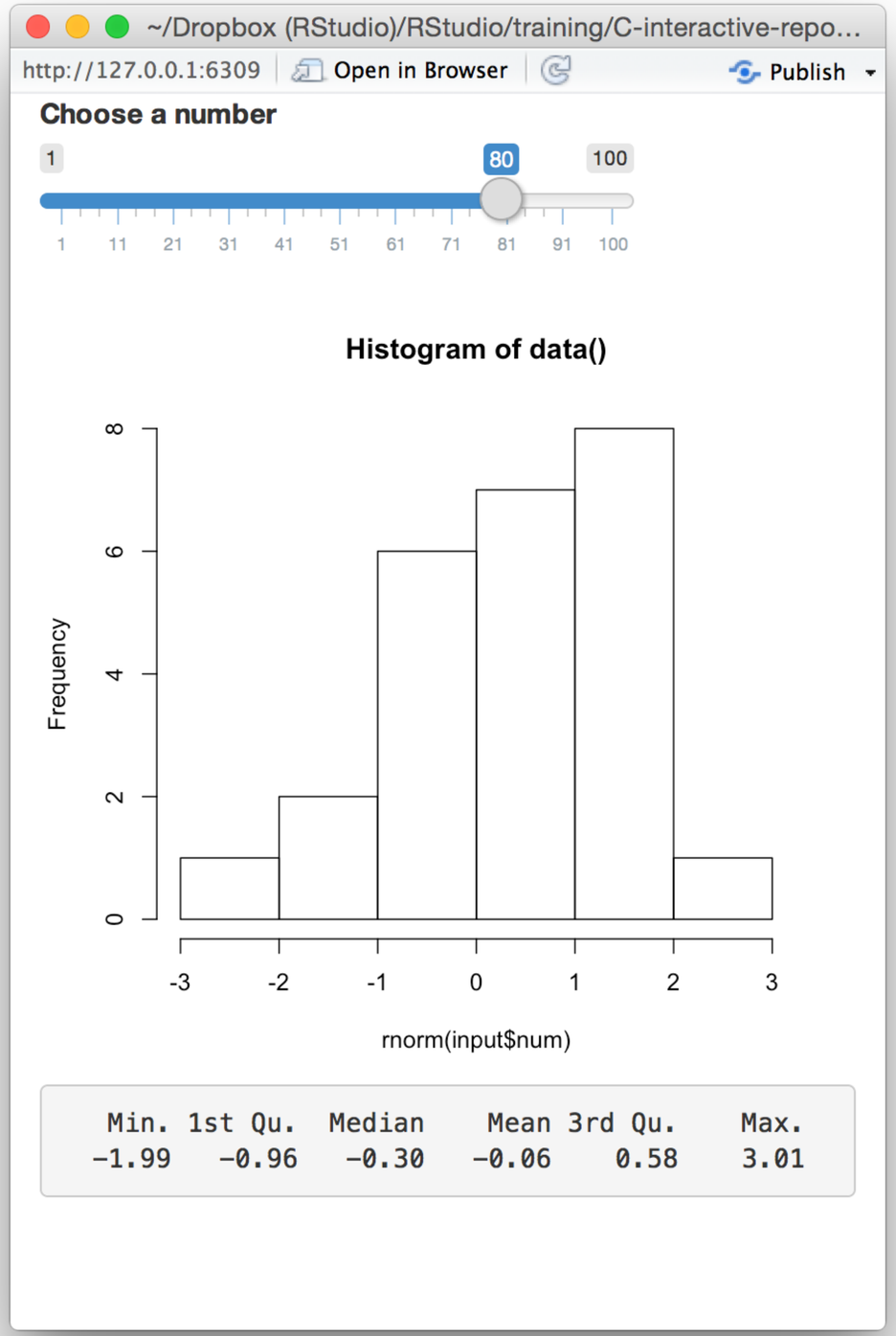


```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

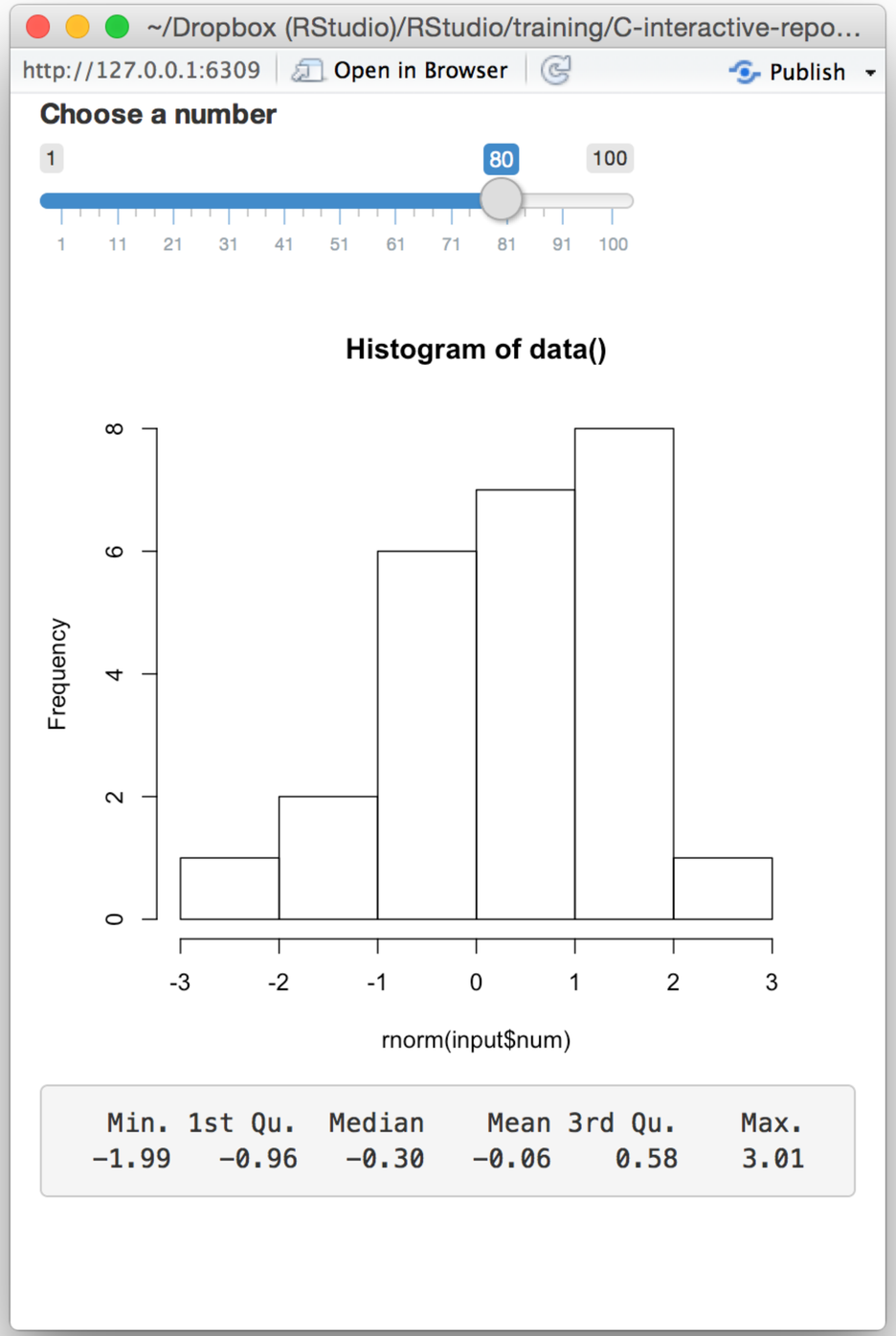


```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```



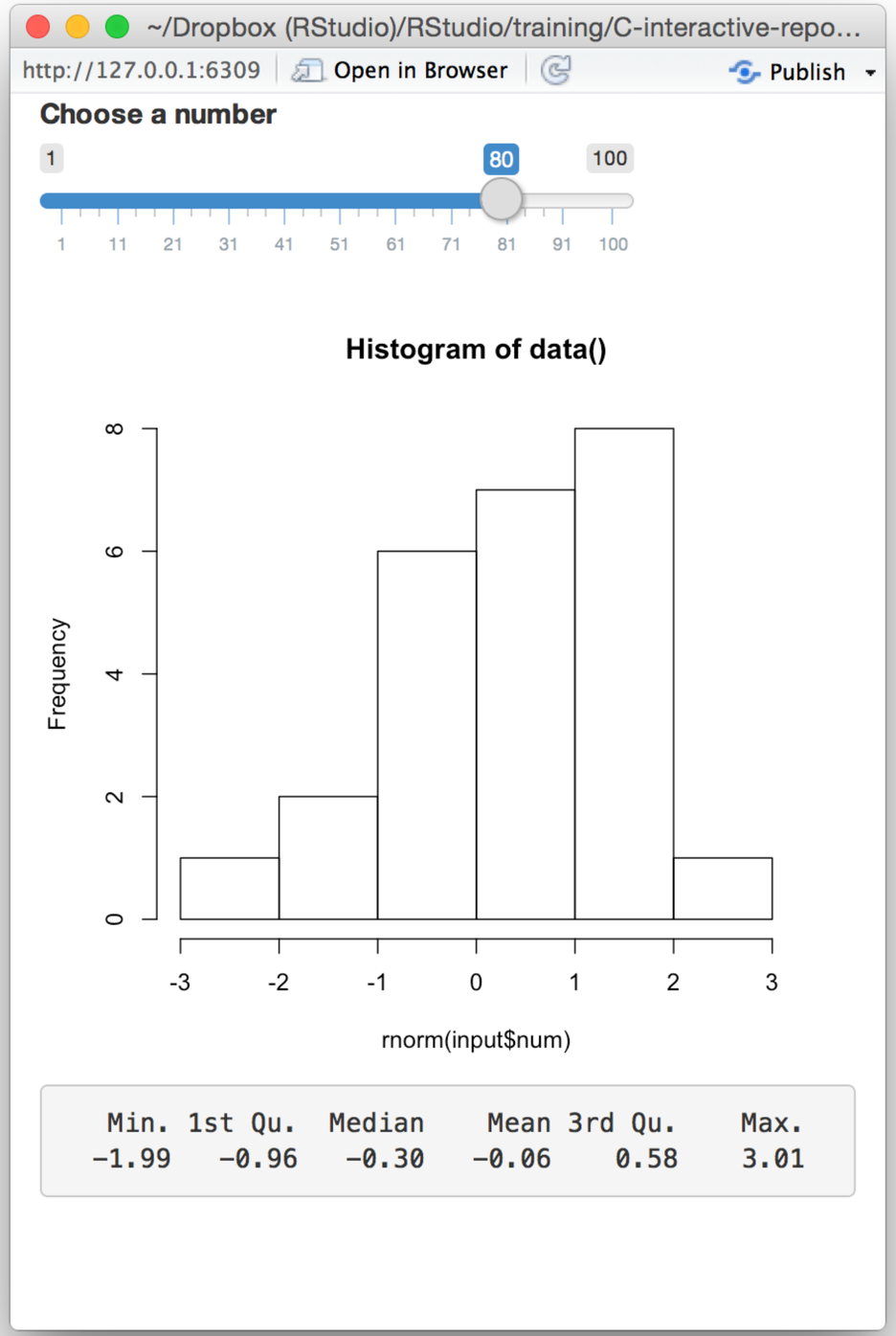
```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

```
> hist(data())))
```



```
input$num
```

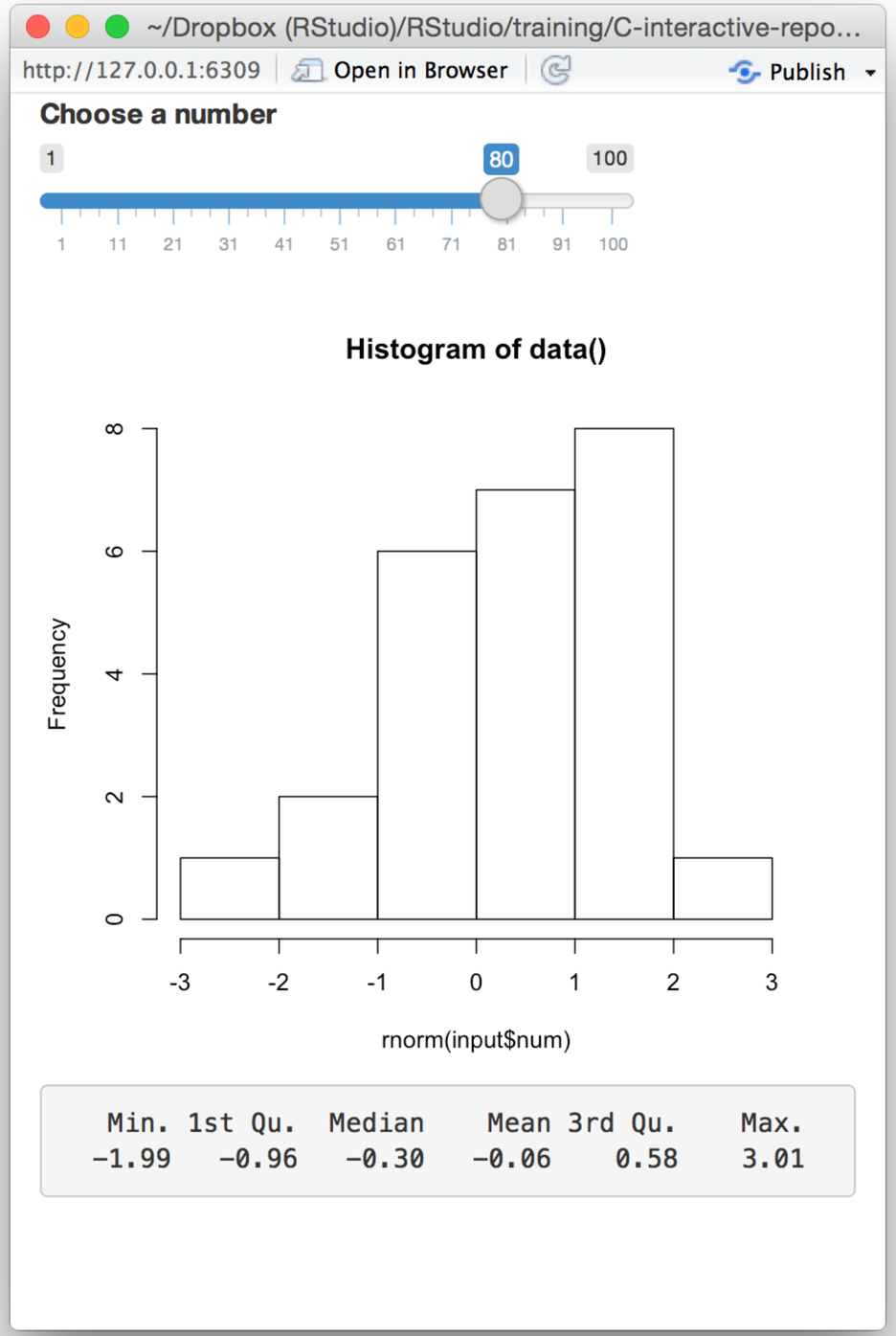
```
data <- reactive({
  rnorm(input$num)
})
```

```
> rnorm(input$num)
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```



```
input$num
```

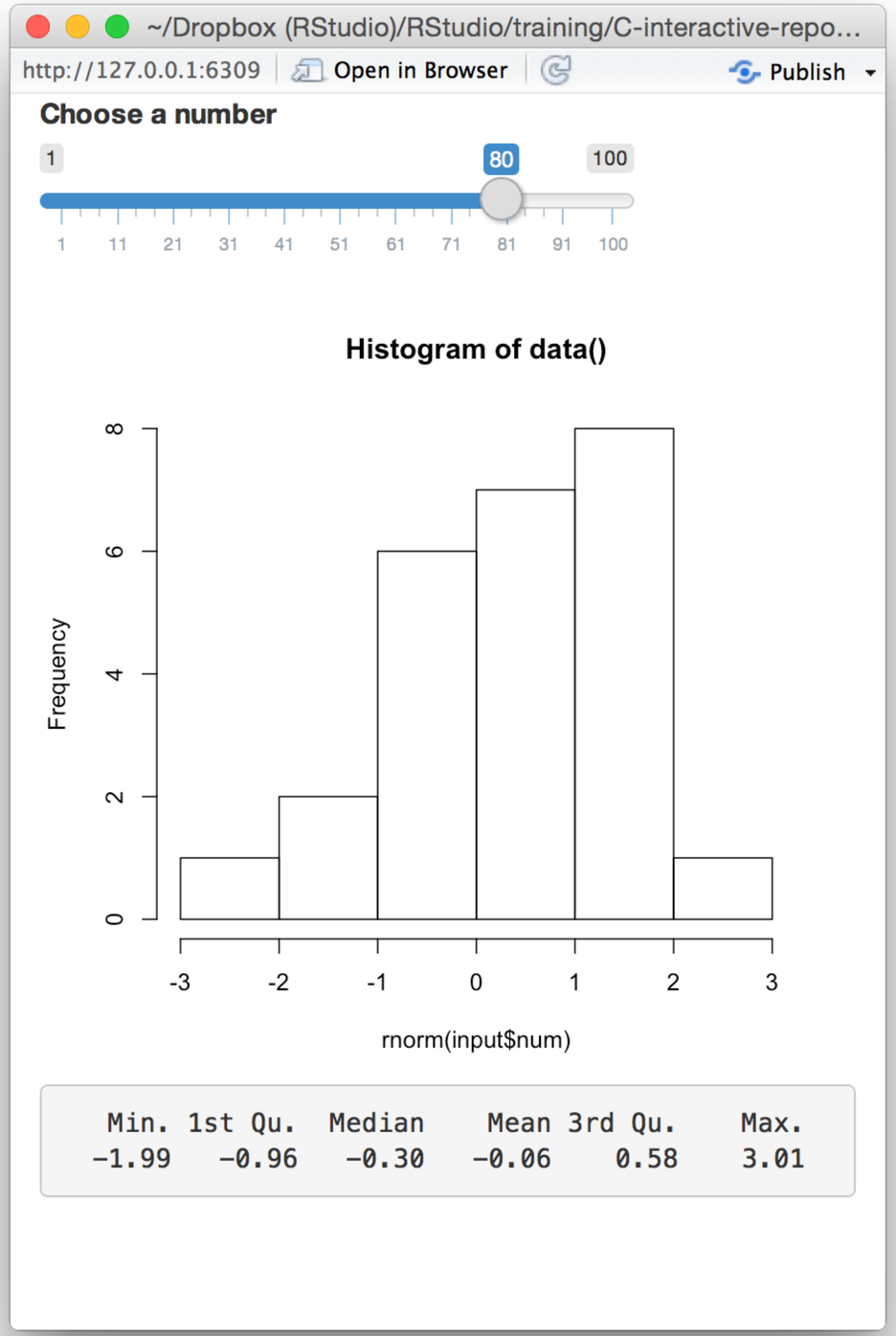
```
data <- reactive({
  rnorm(input$num)
})
```

```
> rnorm(input$num)
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```



```
input$num
```

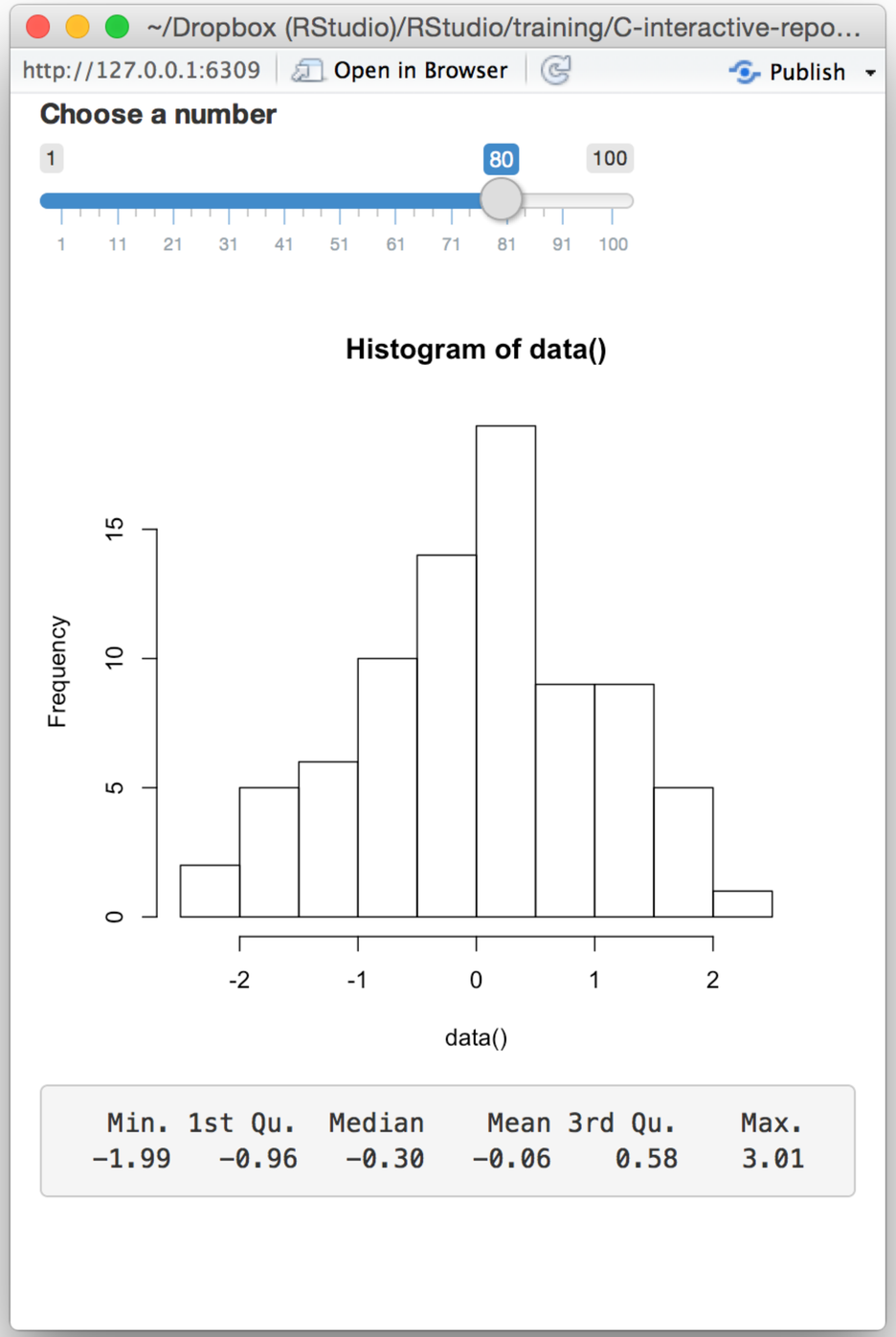
```
data <- reactive({
  rnorm(input$num)
})
```

```
> rnorm(input$num)
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```



```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

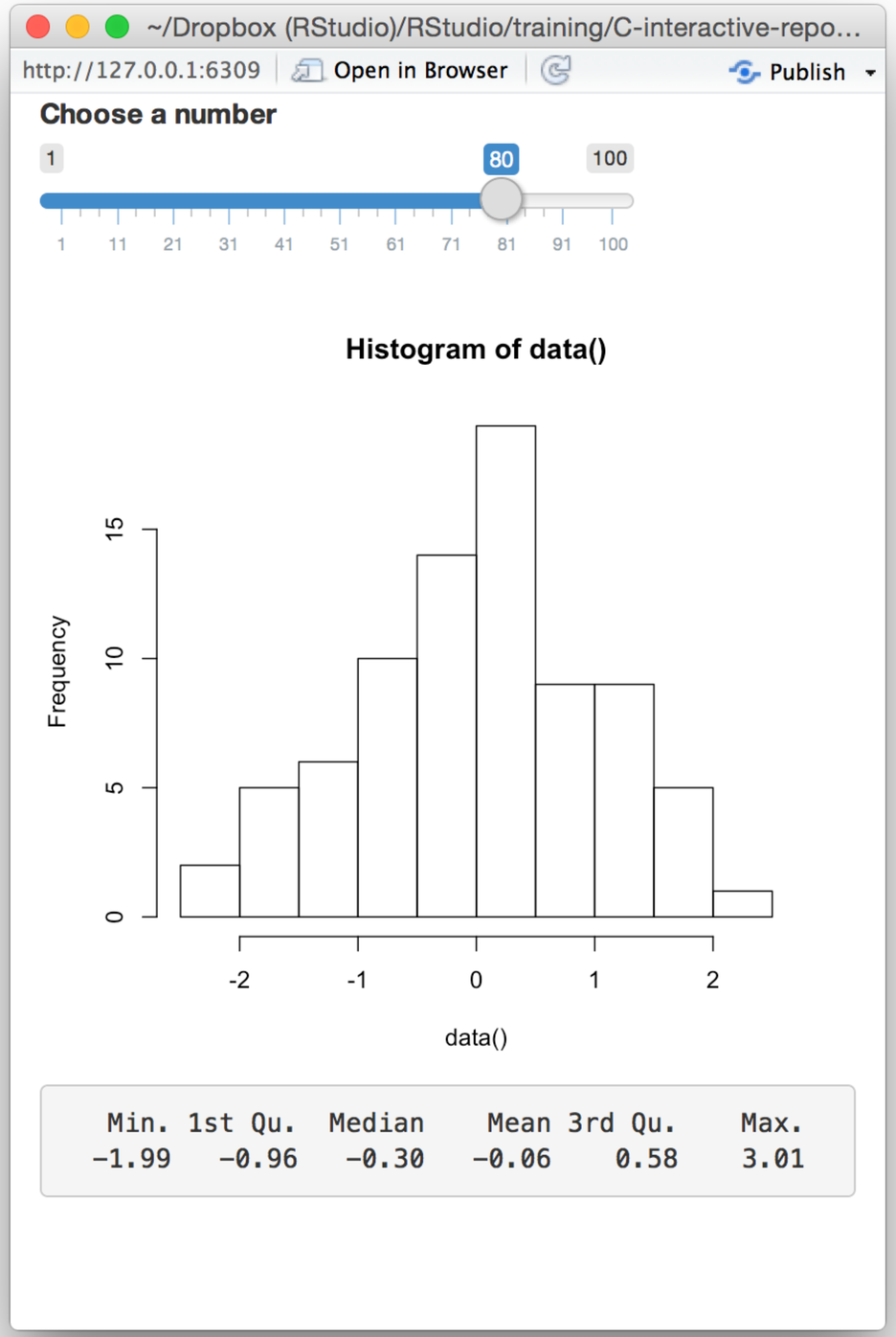
```
> rnorm(input$num)
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```

```
> summary(data())
```



```
input$num
```

```
data <- reactive({
  rnorm(input$num)
})
```

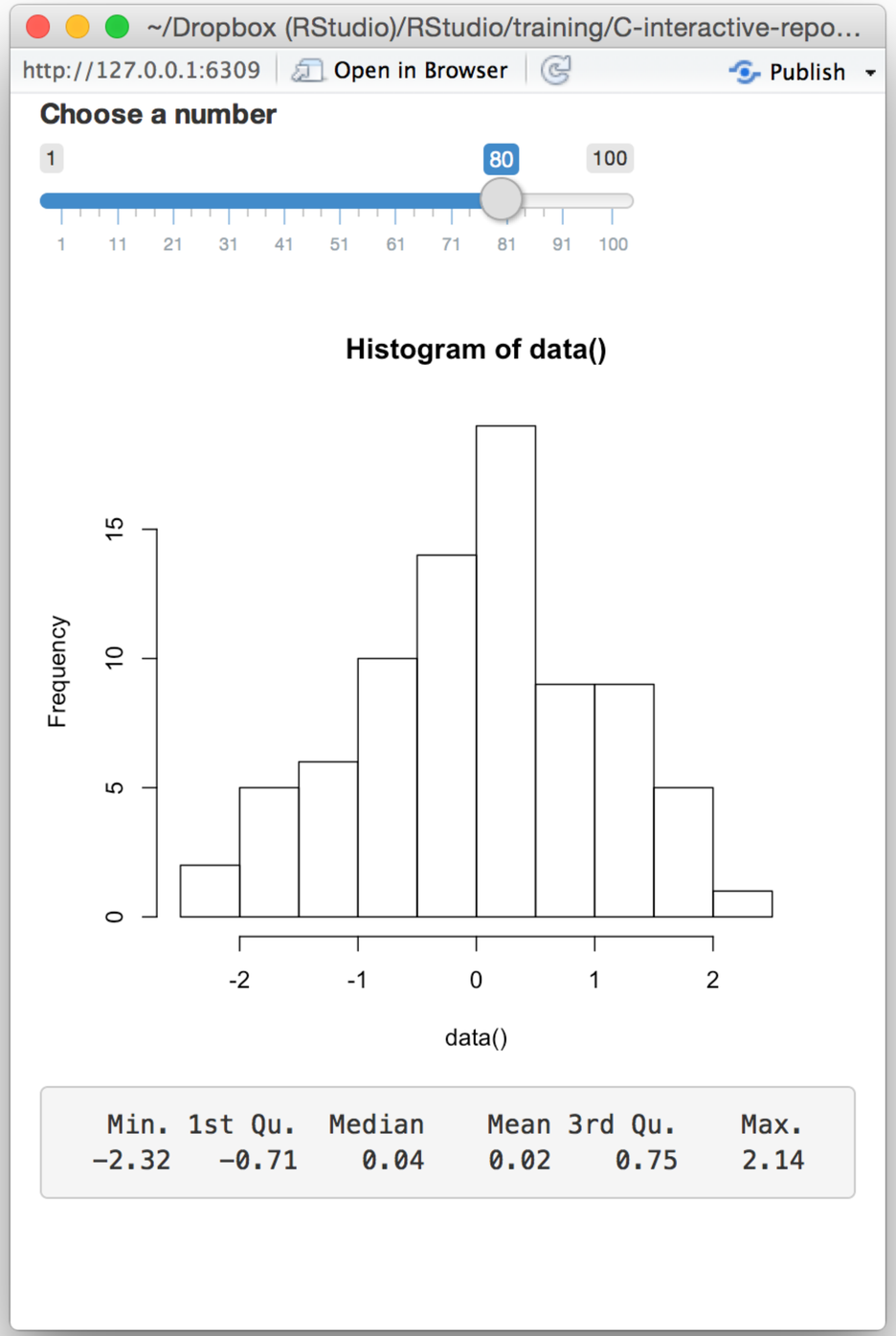
```
> rnorm(input$num)
```

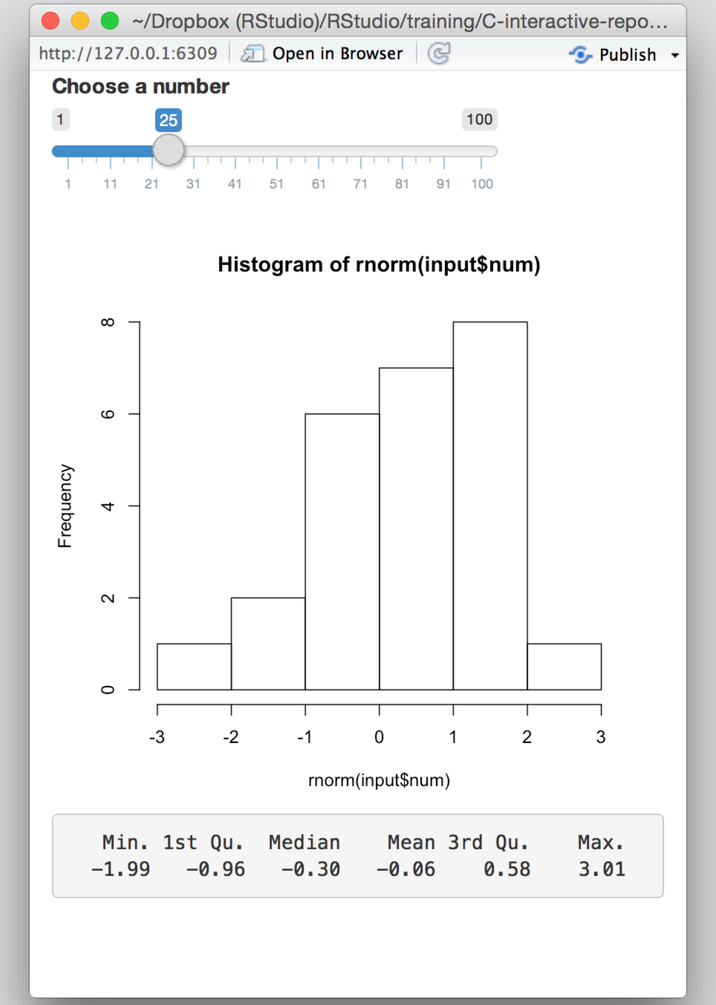
```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
> hist(data())))
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```

```
> summary(data())
```



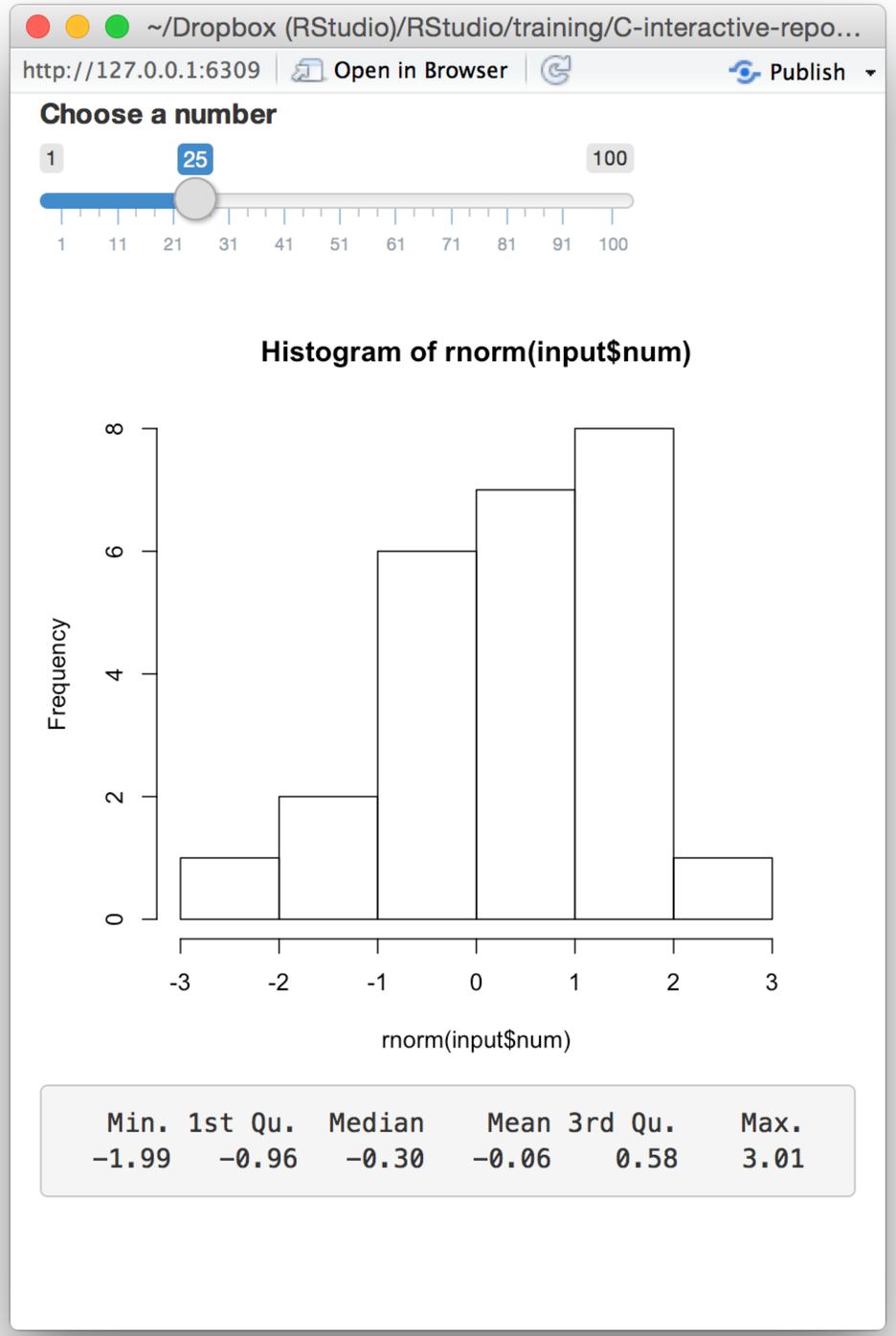


Your Turn

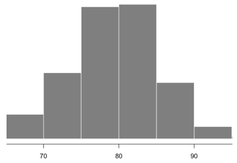
Use **reactive()** to pass the same data to the histogram and the summary.

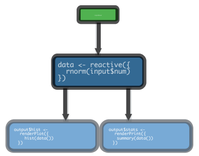
Ensure that you can predict how the app will work.

```
ui <- fluidPage(
  sliderInput("num", “Choose a #”, 1, 100, 50),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   data <- reactive({ rnorm(input$num) })
   output$hist <- renderPlot({
     hist(data())
   })
   output$sum <- renderPrint({
     summary(data())
   })
}
shinyApp(ui = ui, server = server)
```



Use…



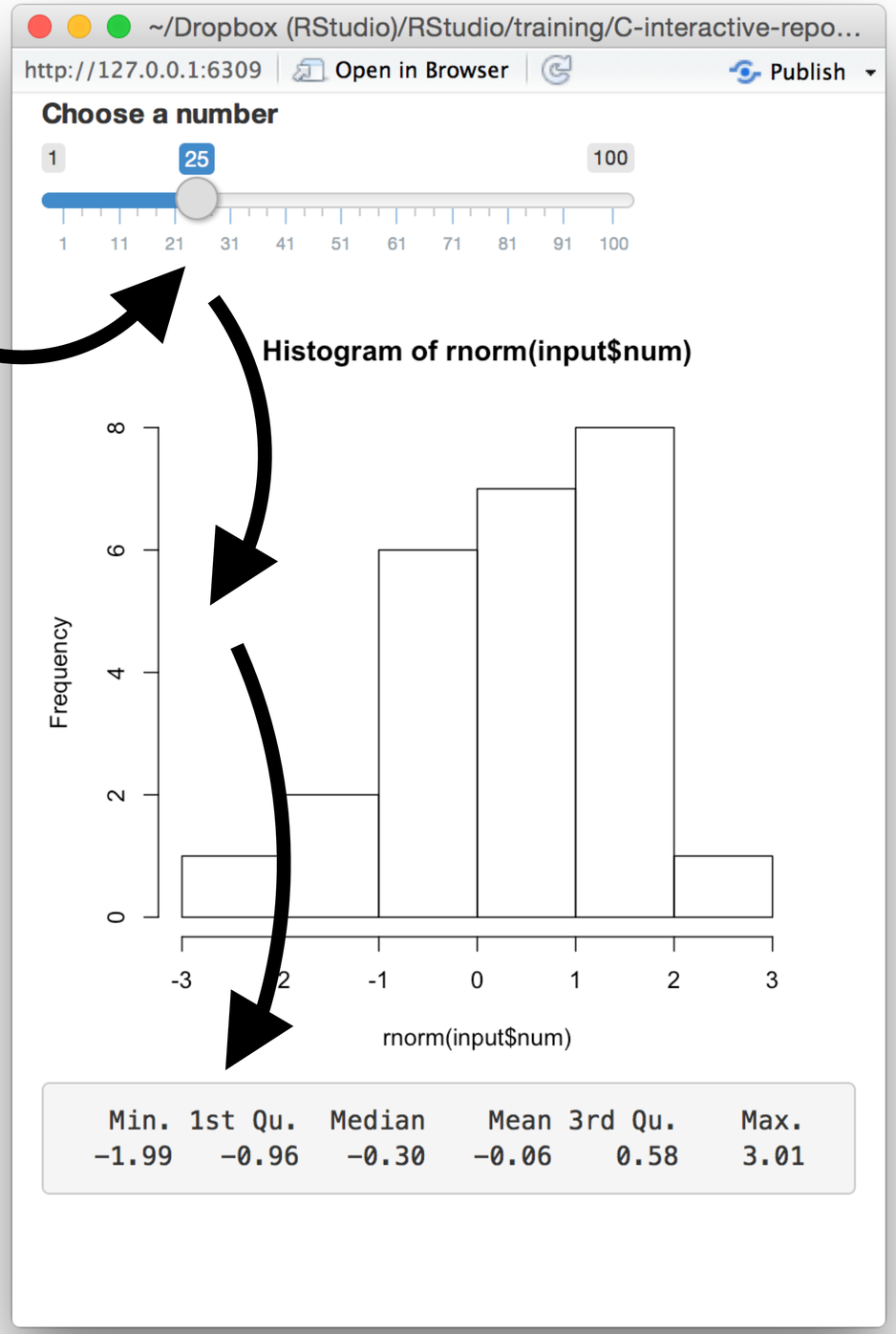


render() to make an **object to display** in the UI.

reactive() to make an **object to use** in downstream code.

```
ui <- fluidPage(
  sliderInput("num", “Choose a #”, 1, 100, 50),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
  data <- reactive({ rnorm(input$num) })
  output$hist <- renderPlot({
    hist(data())
  })
  output$sum <- renderPrint({
    summary(data())
  })
}
shinyApp(ui = ui, server = server)
```

Can we delay the reactions?



isolate()

Makes a reactive object non-reactive.

```
renderPlot({ hist(rnorm(isolate(input$num))) })
```

isolate()

Makes a reactive object non-reactive.

```
renderPlot({ hist(rnorm(isolate(input$num))) })
```

Builds an object that: does nothing

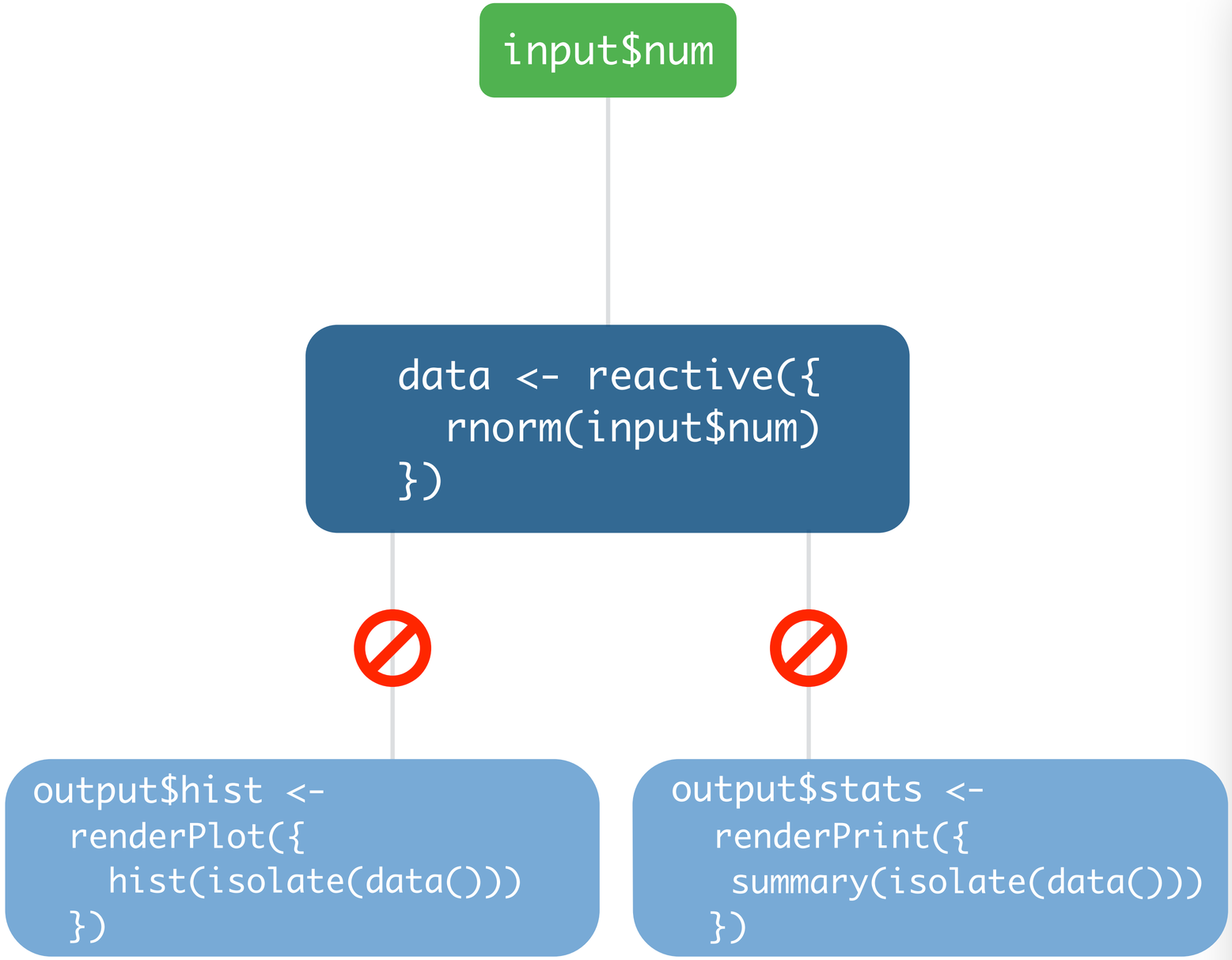
isolate()

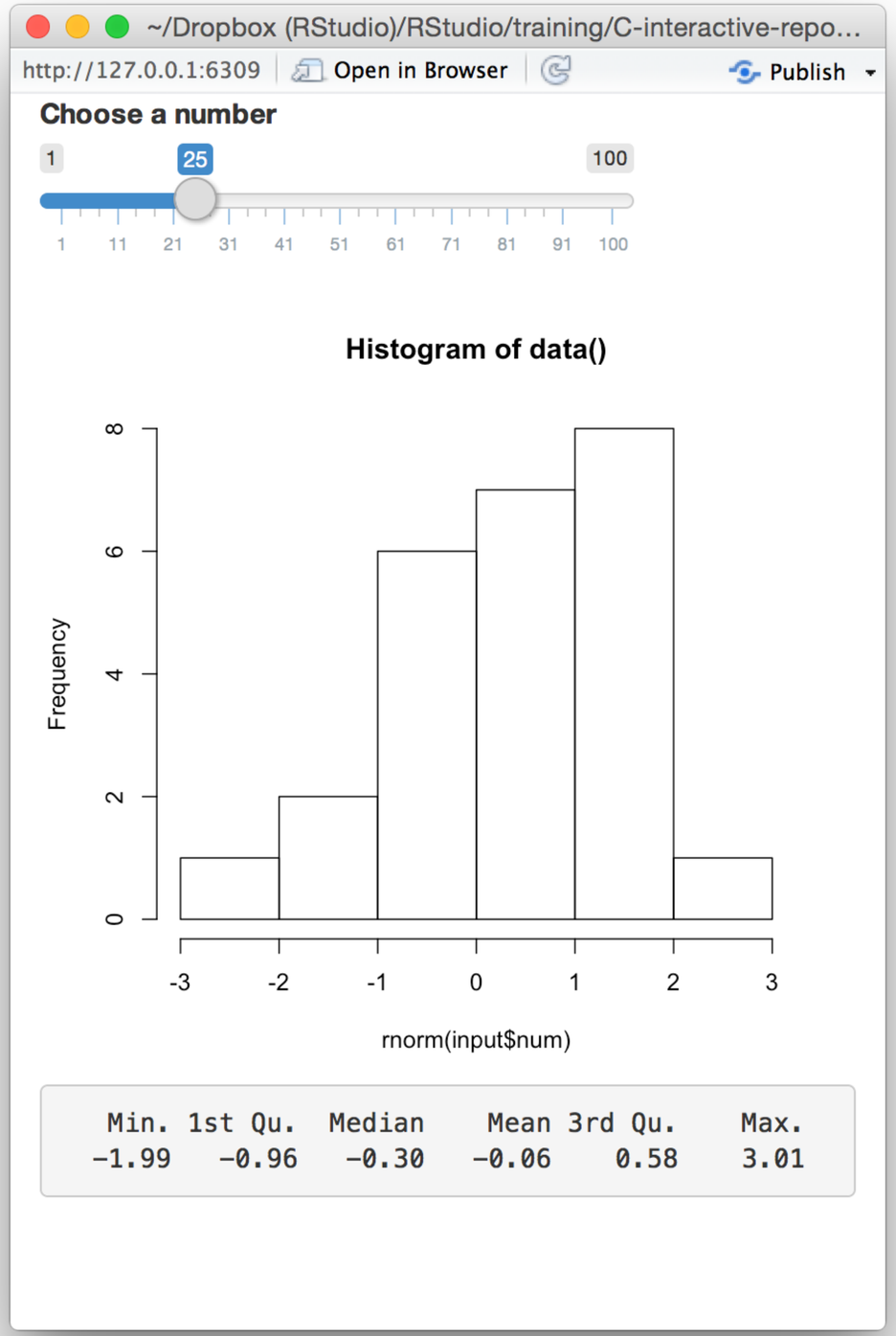
Makes a reactive object non-reactive.

```
renderPlot({ hist(rnorm(isolate(input$num))) })
```

Builds an object that: does nothing

When notified by: any reactive value wrapped by isolate


input$num
data <- reactive({
rnorm(input$num)
})
output$hist <-
renderPlot({
hist(isolate(data()))
})
output$stats <-
renderPrint({
summary(isolate(data()))
})

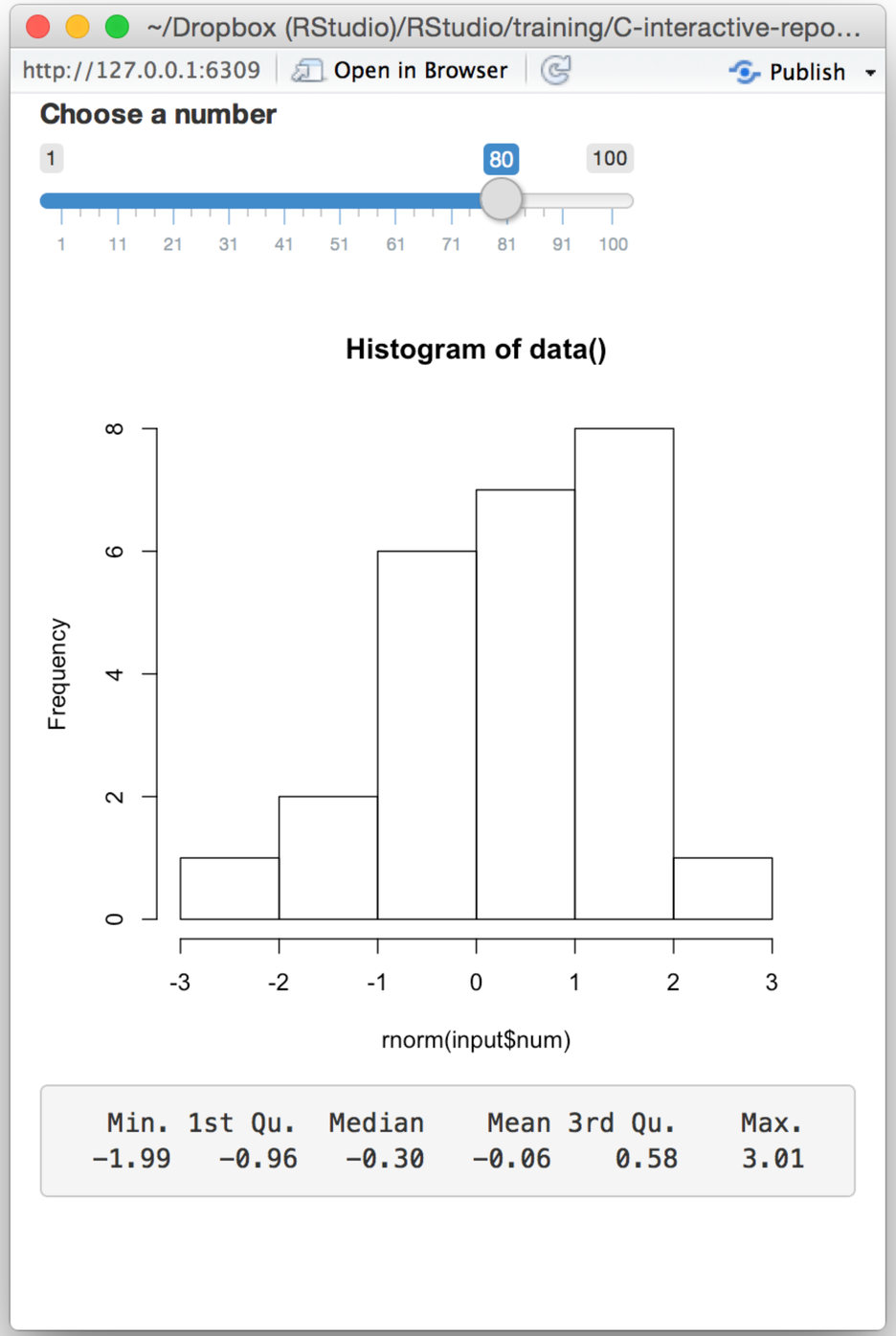

~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
25
100
1 11 21 31 41 51 61 71 81 91 100
Histogram of data()
Frequency
0 2 4 6 8
-3 -2 -1 0 1 2 3
rnorm(input$num)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.99 -0.96 -0.30 -0.06 0.58 3.01

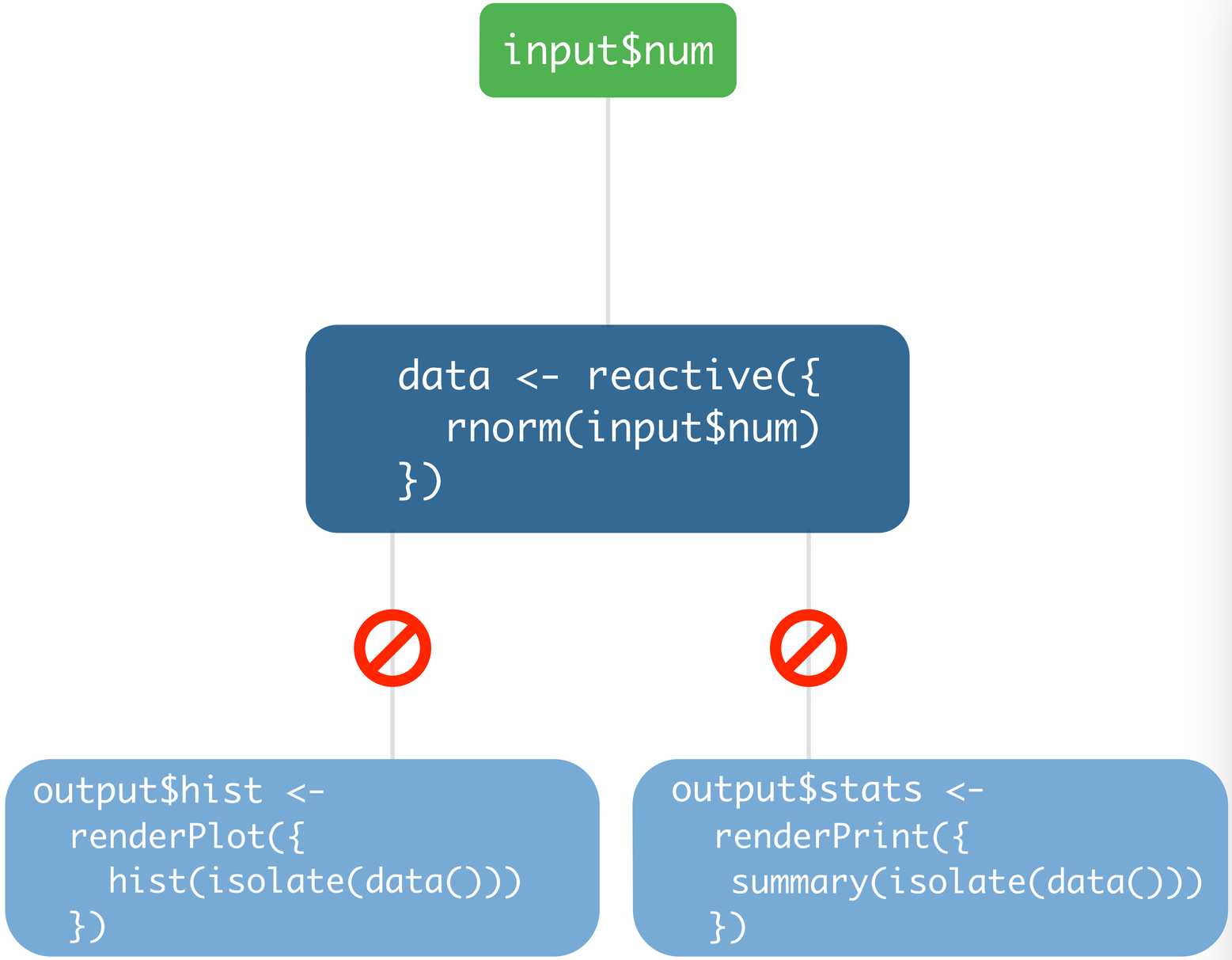
```
input$num
```

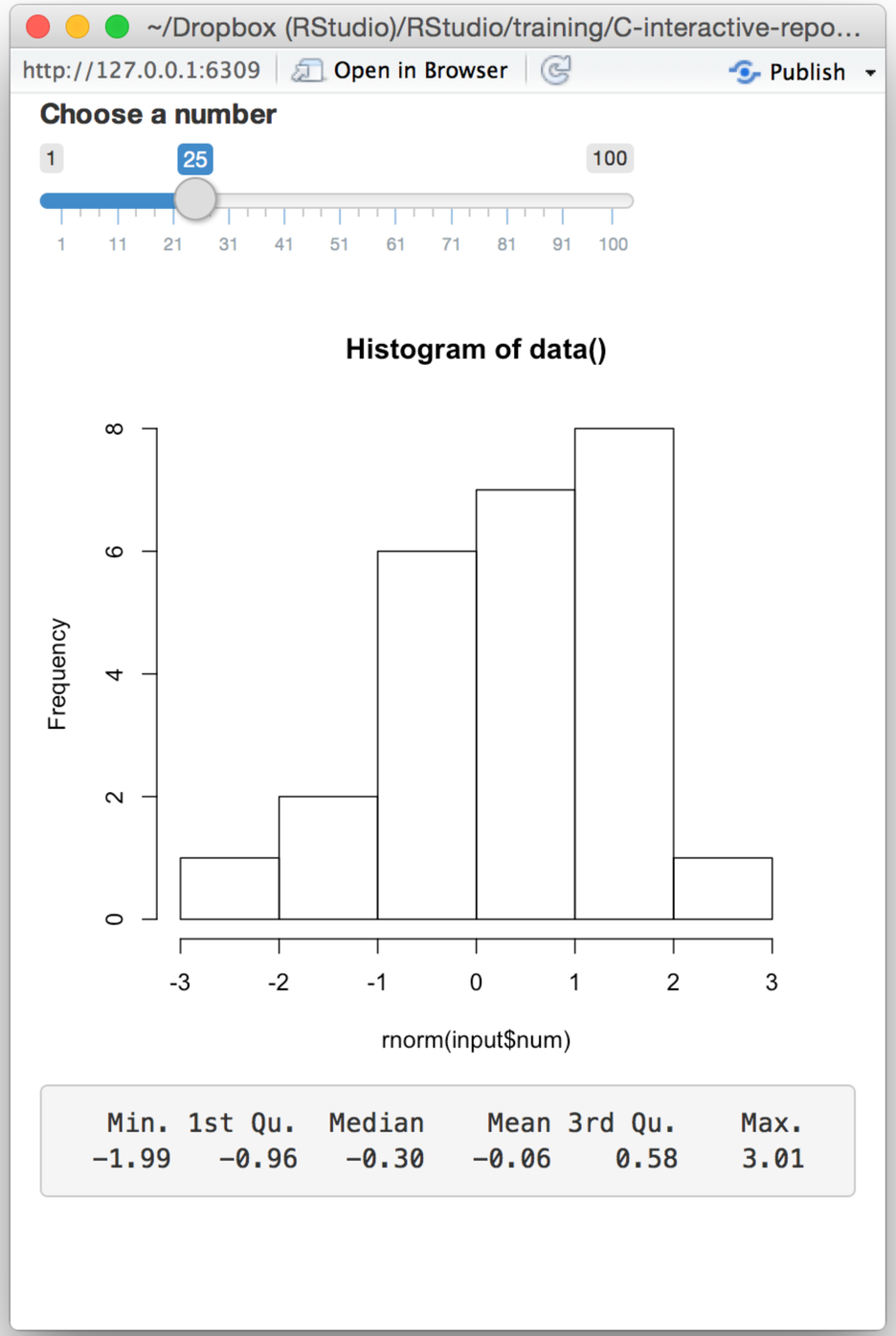
```
data <- reactive({
  rnorm(input$num)
})
```

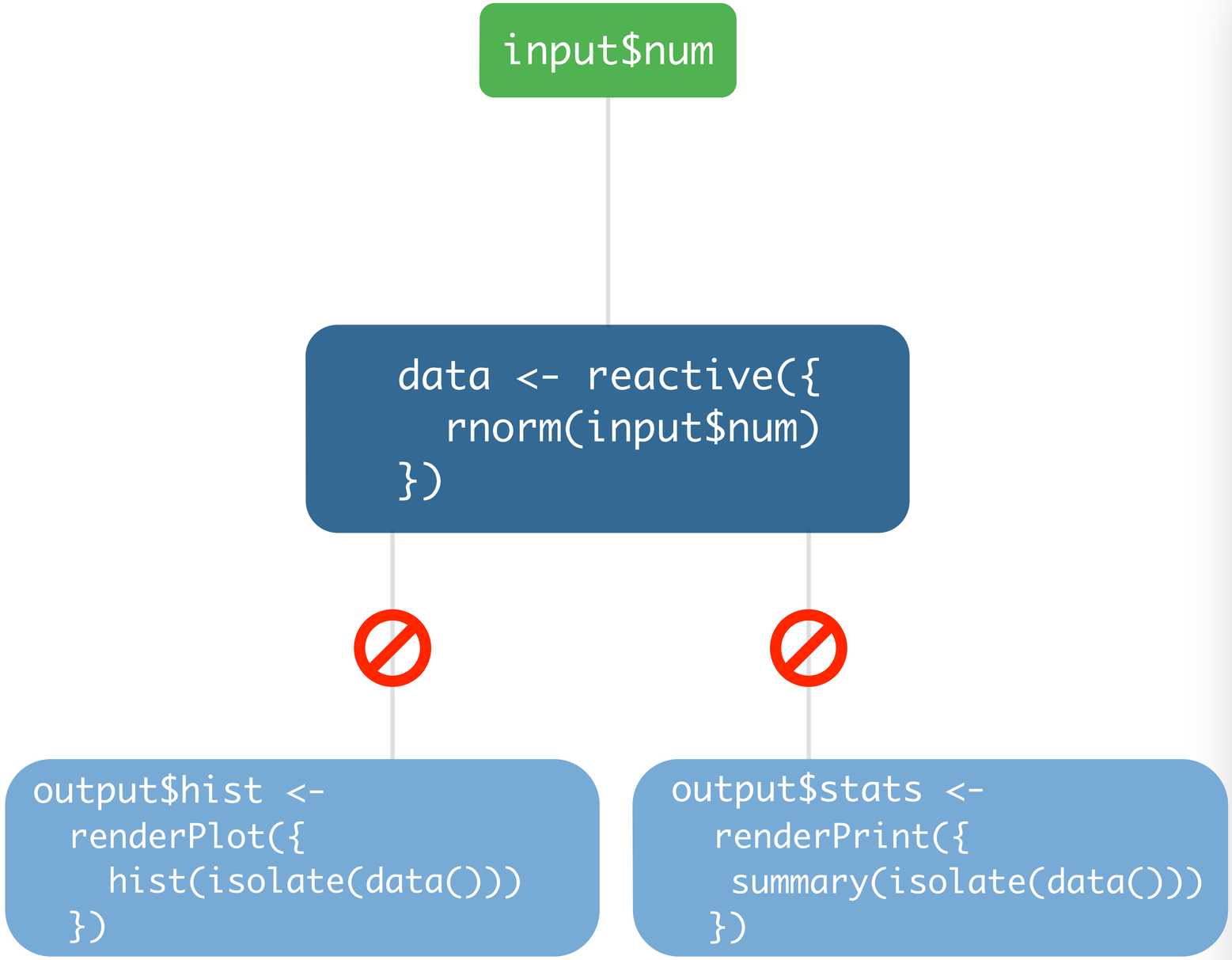
```
output$hist <-
  renderPlot({
    hist(isolate(data()))
  })
```

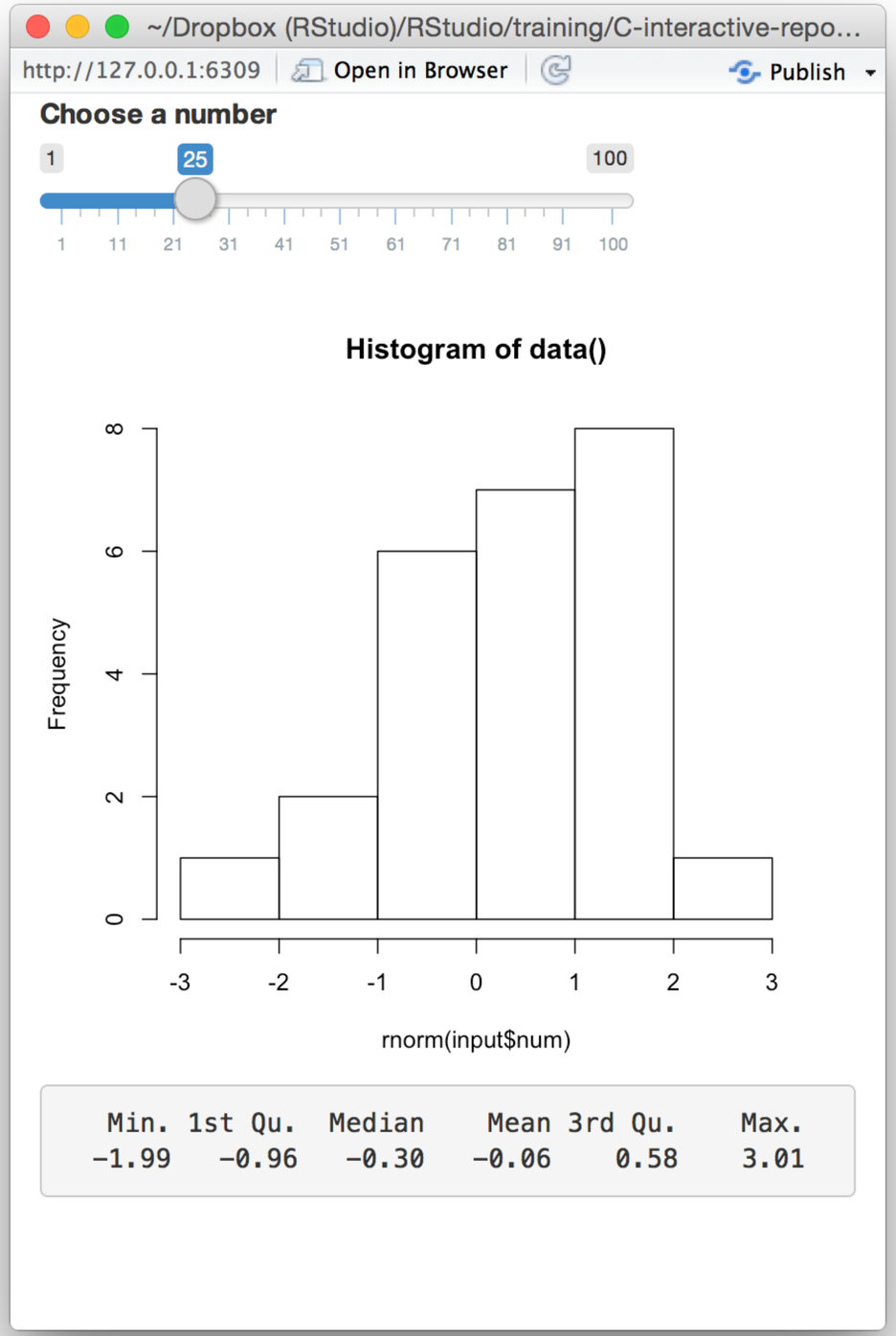
```
output$stats <-
  renderPrint({
    summary(isolate(data()))
  })
```



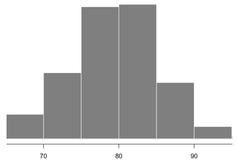

input$num
data <- reactive({
rnorm(input$num)
})
output$hist <-
renderPlot({
hist(isolate(data()))
})
output$stats <-
renderPrint({
summary(isolate(data()))
})


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
25
100
1 11 21 31 41 51 61 71 81 91 100
Histogram of data()
Frequency
0 2 4 6 8
-3 -2 -1 0 1 2 3
rnorm(input$num)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.99 -0.96 -0.30 -0.06 0.58 3.01

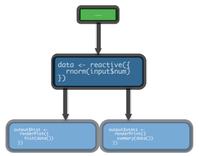

input$num
data <- reactive({
rnorm(input$num)
})
output$hist <-
renderPlot({
hist(isolate(data()))
})
output$stats <-
renderPrint({
summary(isolate(data()))
})


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
25
100
1 11 21 31 41 51 61 71 81 91 100
Histogram of data()
Frequency
0 2 4 6 8
-3 -2 -1 0 1 2 3
rnorm(input$num)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.99 -0.96 -0.30 -0.06 0.58 3.01

Use…



render() to make an **object to display** in the UI.



reactive() to make an **object to use** in downstream code.

isolate() to return a **non-reactive object**.

eventReactive()

Let's you control when an expression is invalidated

```
data <- eventReactive(input$go, { rnorm(input$num) })
```

eventReactive()

Let's you control when an expression is invalidated

```
data <- eventReactive(input$go, { rnorm(input$num) })
```

Builds an object that:

notifies objects that use it that they are invalid

eventReactive()

Let's you control when an expression is invalidated

```
data <- eventReactive(input$go, { rnorm(input$num) })
```

Builds an object that:

notifies objects that use it that they are invalid

When notified by:

this or these reactive value(s) **and no others**

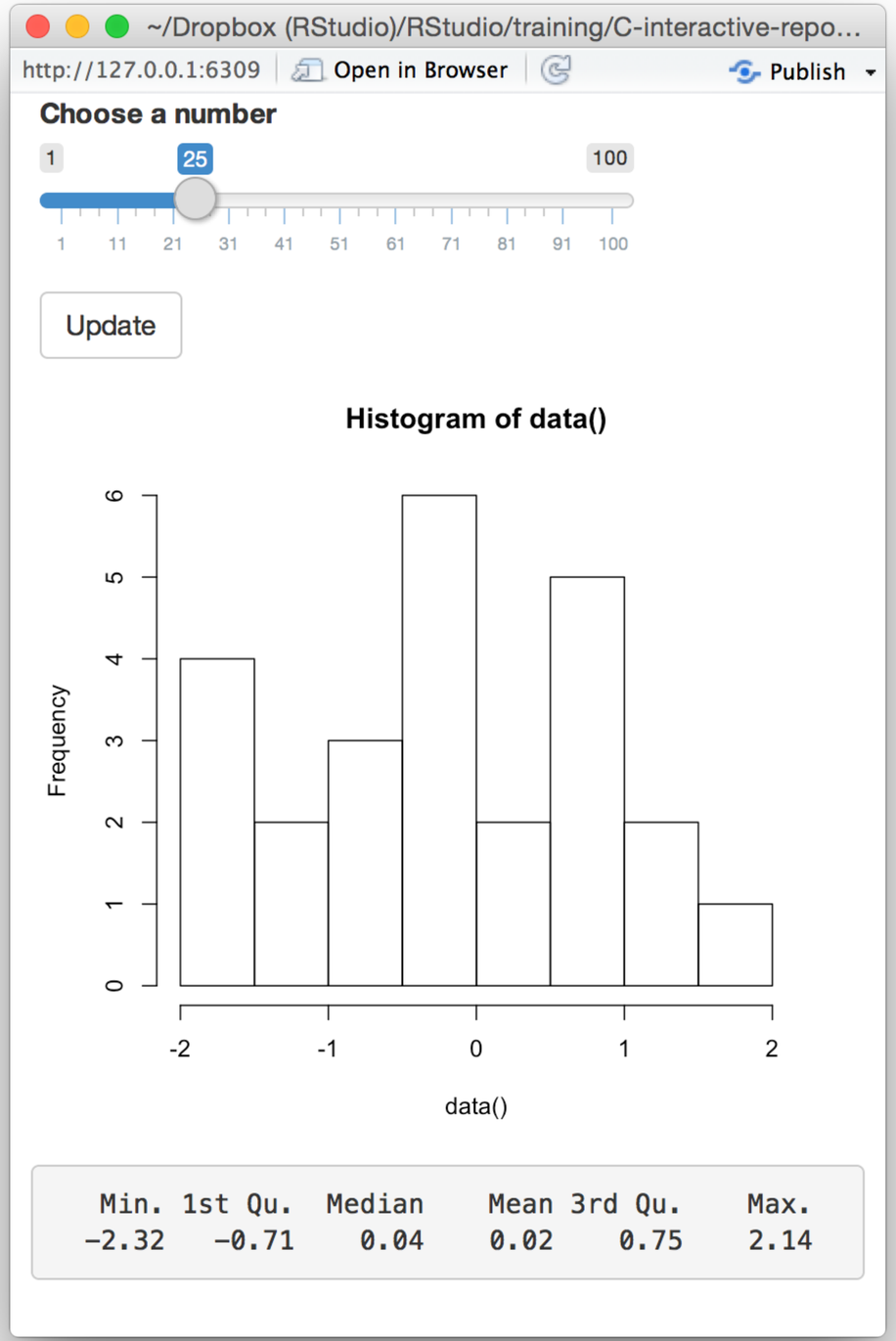
input$num

input$go

```
data <- eventReactive(input$go, {
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```



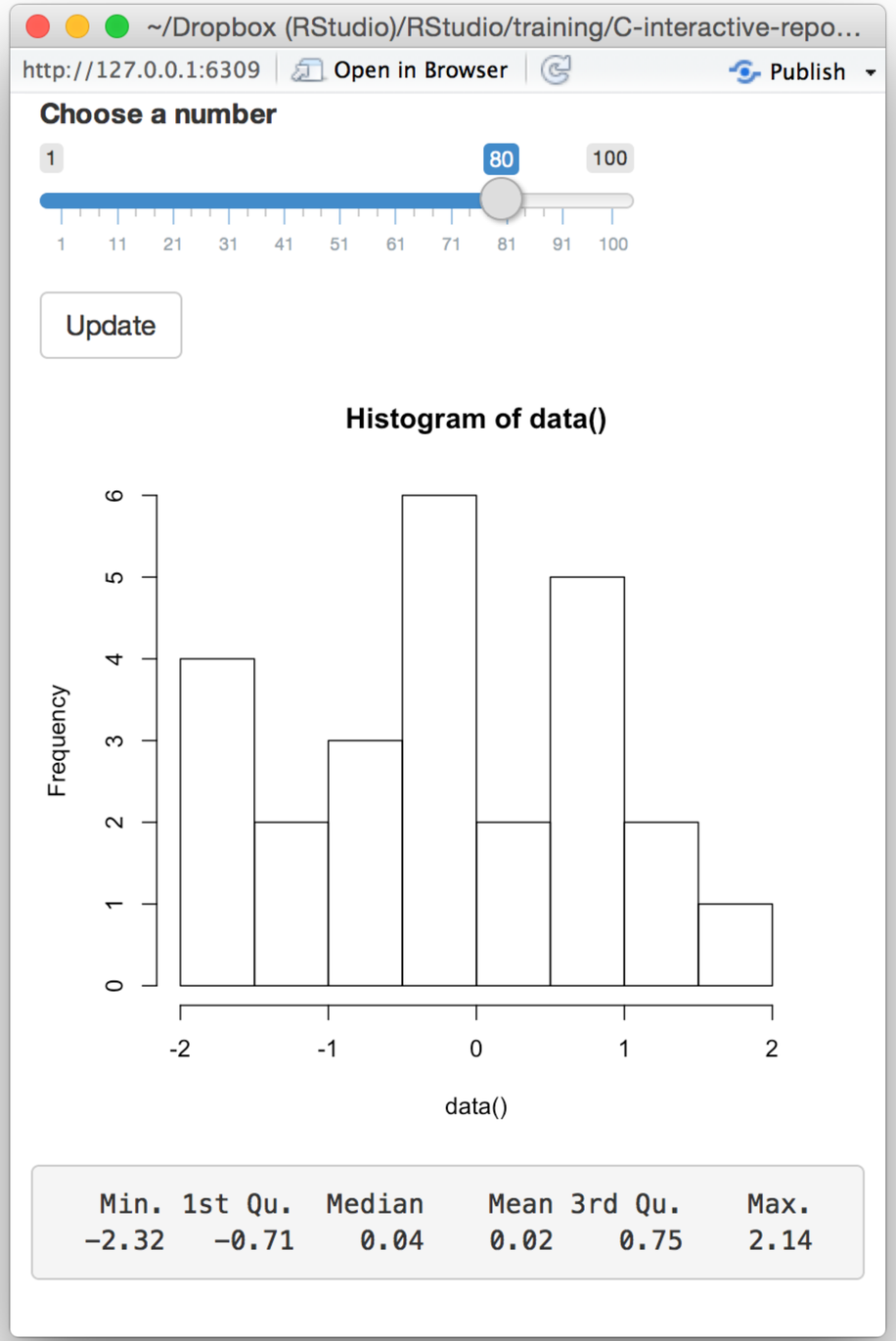
input$num

input$go

```
data <- eventReactive(input$go, {
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```



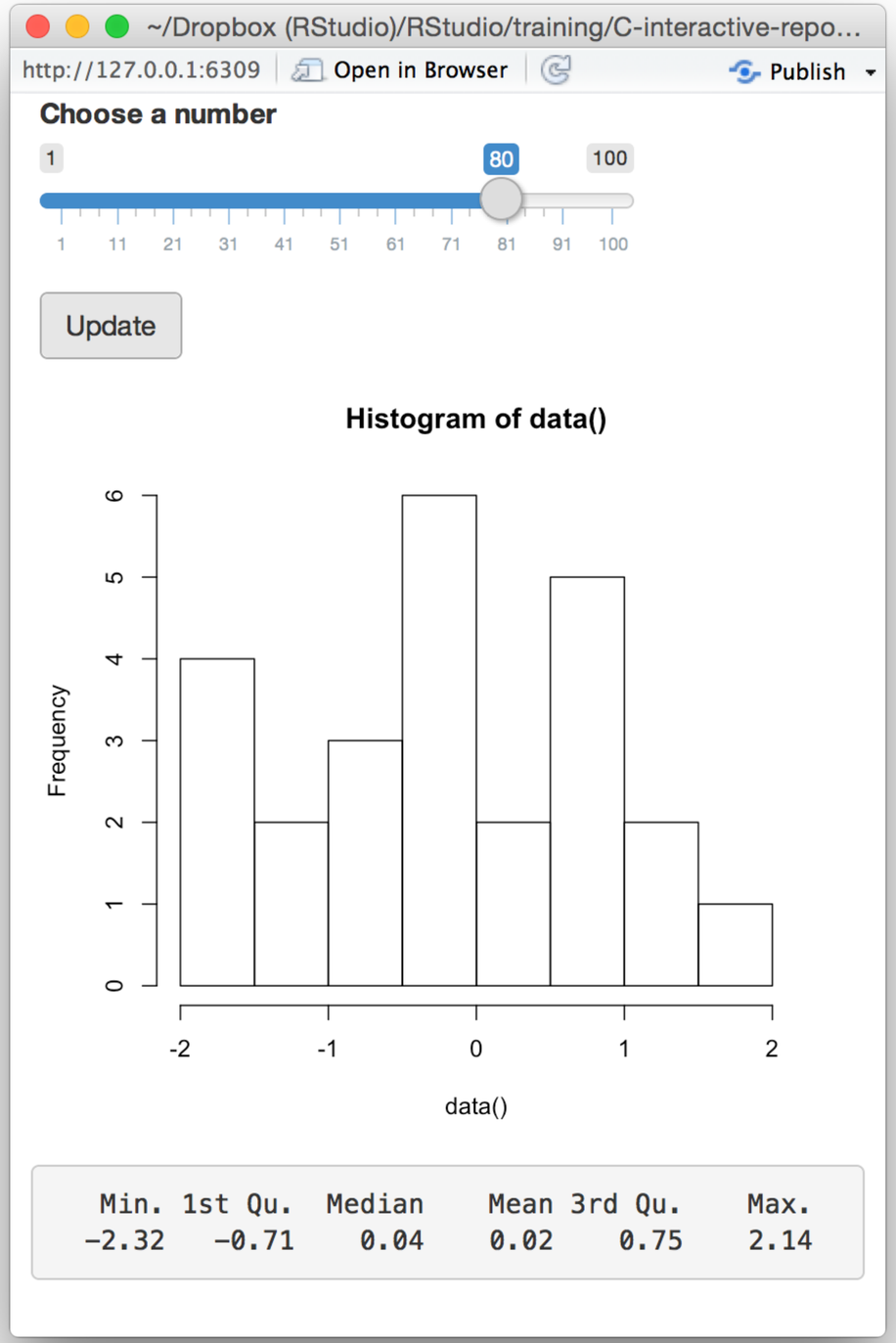
input$num

input$go

```
data <- eventReactive(input$go, {
  rnorm(input$num)
})
```

```
output$hist <-
  renderPlot({
    hist(data())
  })
```

```
output$stats <-
  renderPrint({
    summary(data())
  })
```



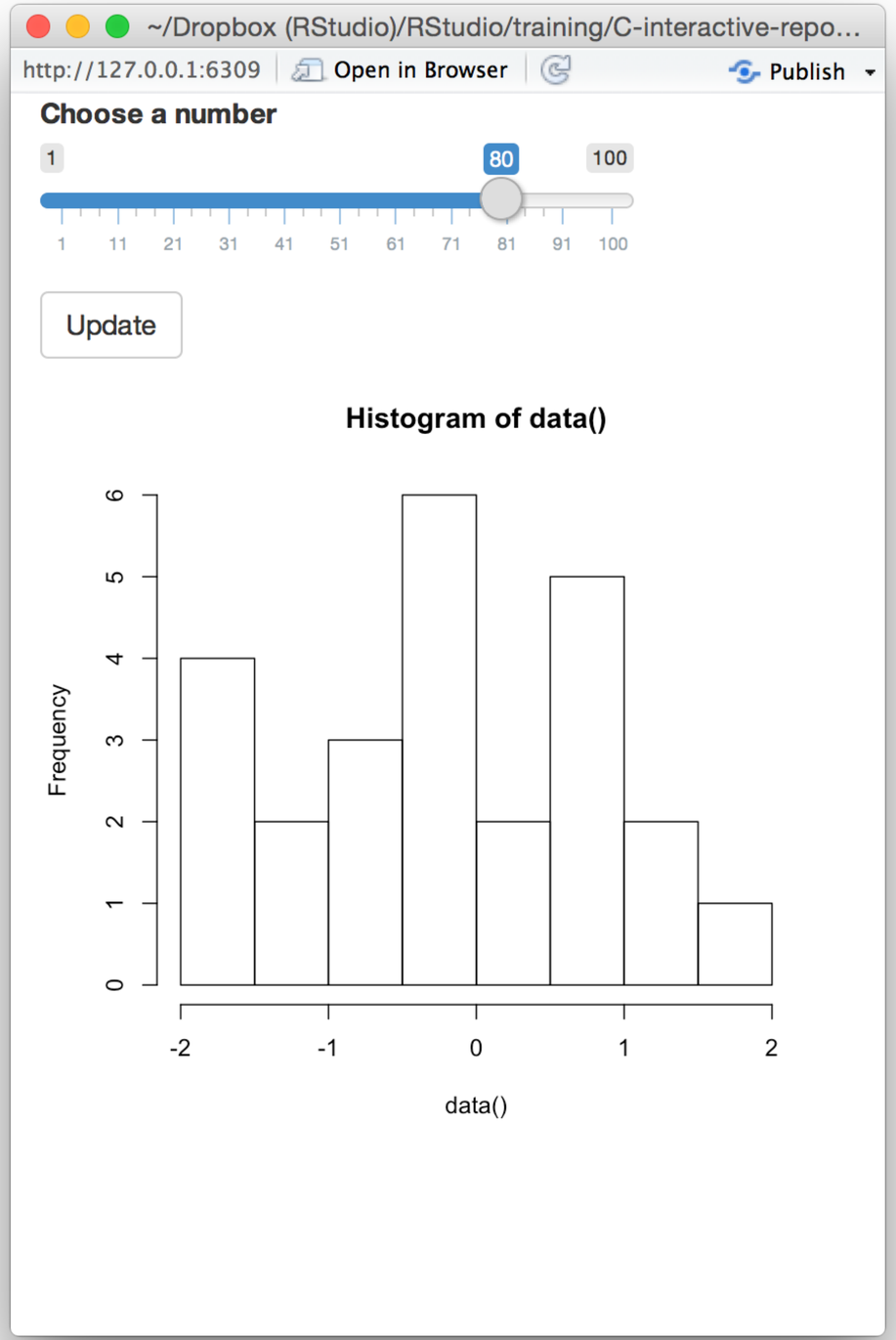
input$num

input$go

```
data <- eventReactive(input$go, {
  rnorm(input$num)
})
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$num

input$go

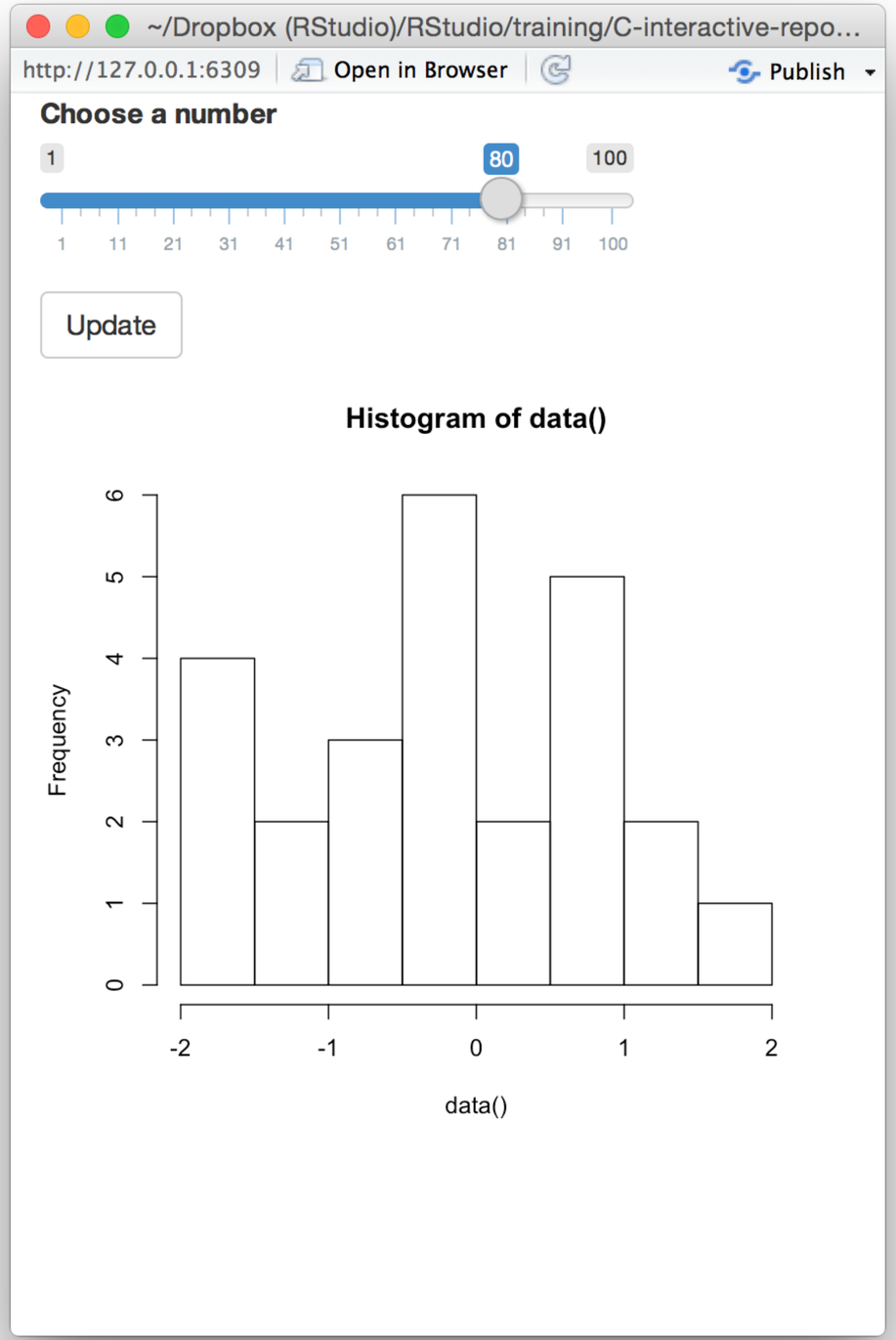
```
data <- eventReactive(input$go, {
  rnorm(input$num)
})
```

```
> rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
     summary(data())
  })
```

```
> hist(data()))
```



input$num

input$go

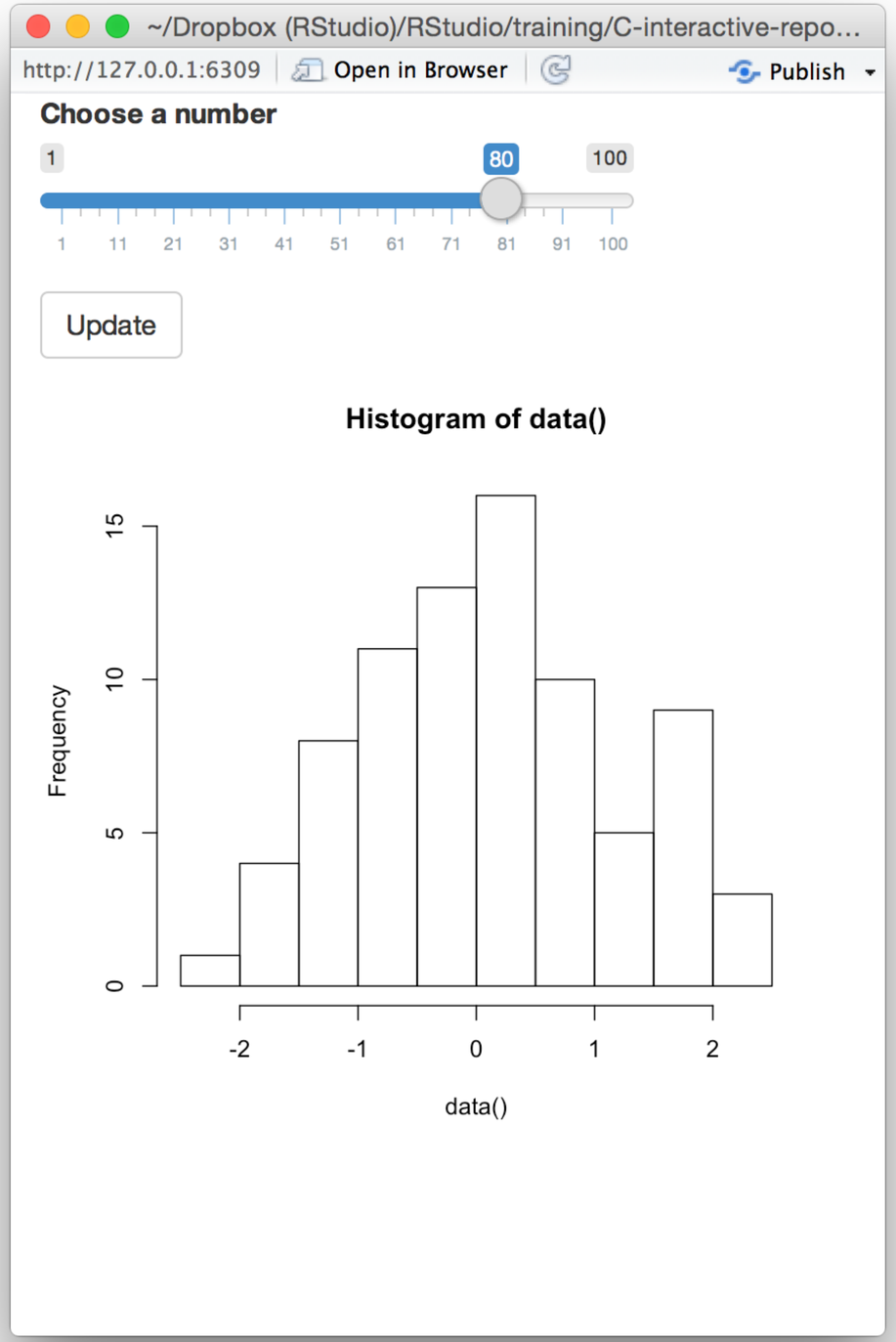
```
data <- eventReactive(input$go, {
  rnorm(input$num)
})
```

```
> rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```



input$num

input$go

```
data <- eventReactive(input$go, {
  rnorm(input$num)
})
```

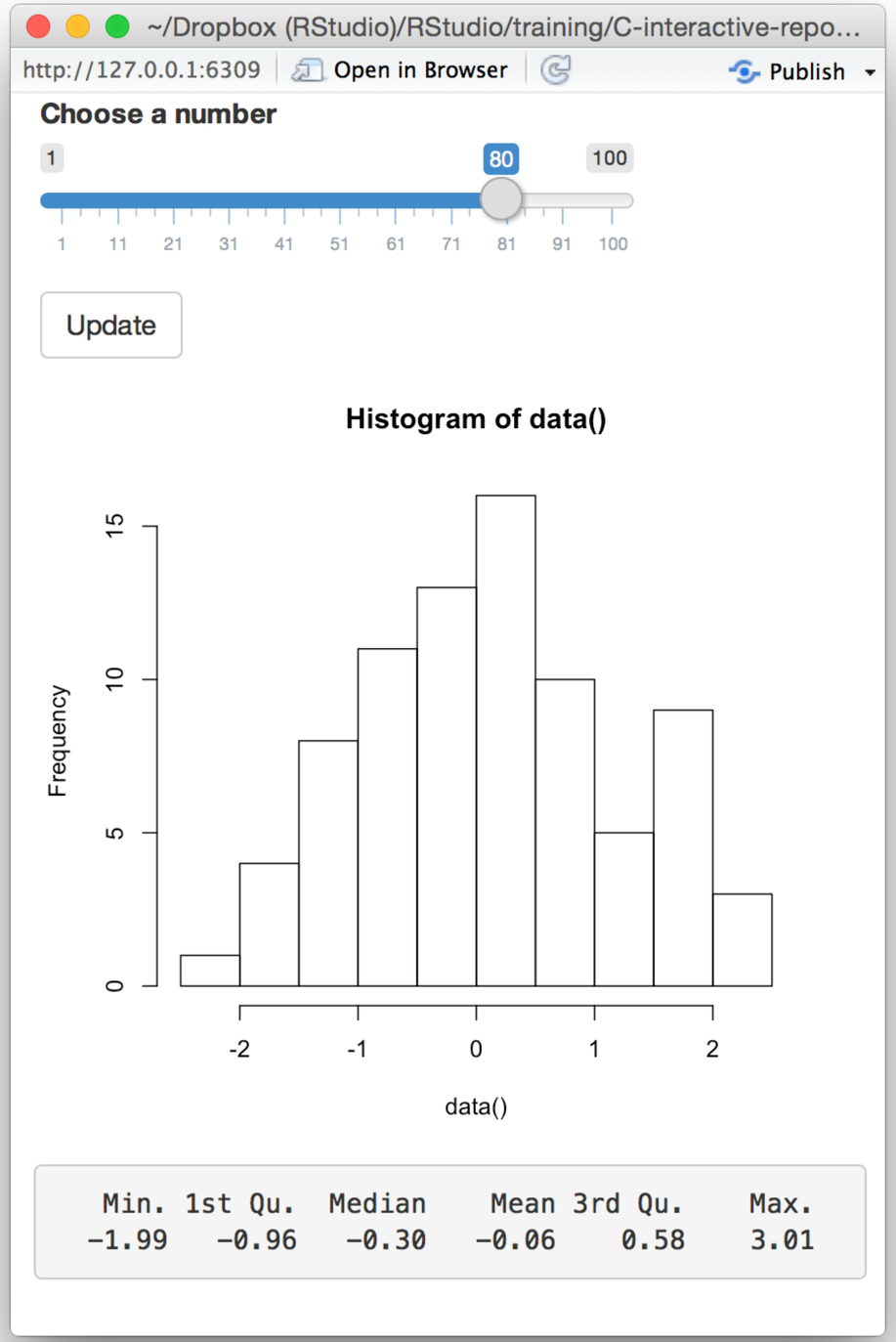
```
> rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

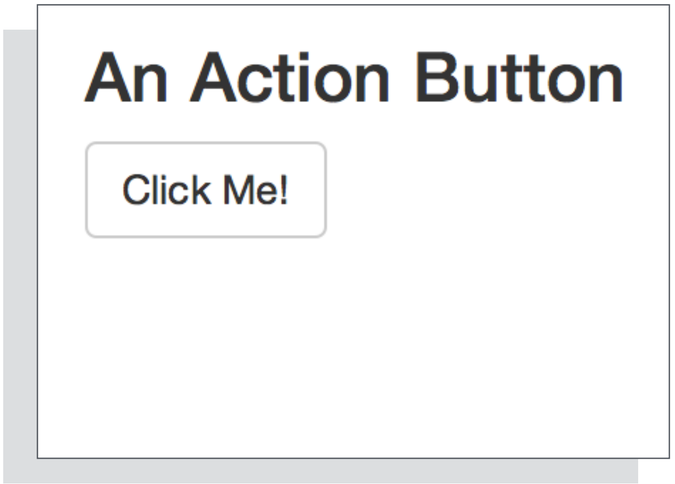
```
output$stats <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```

```
> summary(data())
```



Action buttons




```
actionButton(inputId = "go", label = "Click Me!")
```

The value of an action button increases by one each time it is pressed.

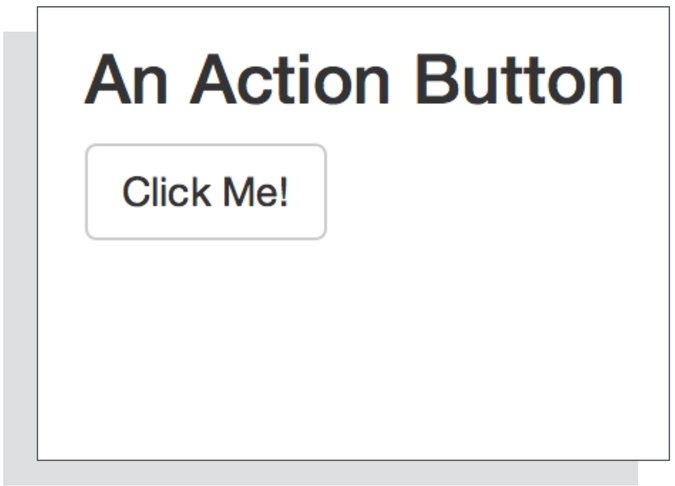
Action buttons

input function

```
actionButton(inputId = "go", label = "Click Me!")
```

The value of an action button increases by one each time it is pressed.

Action buttons



input function

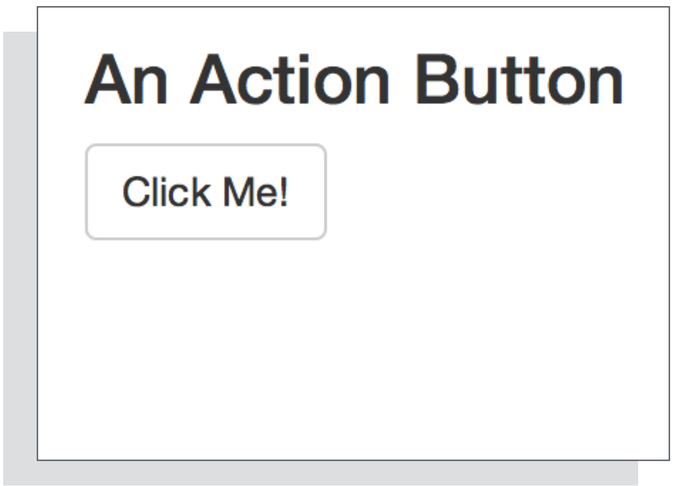
Notice: Id not ID

input name (for internal use)

```
actionButton(inputId = "go", label = "Click Me!")
```

The value of an action button increases by one each time it is pressed.

Action buttons



input function

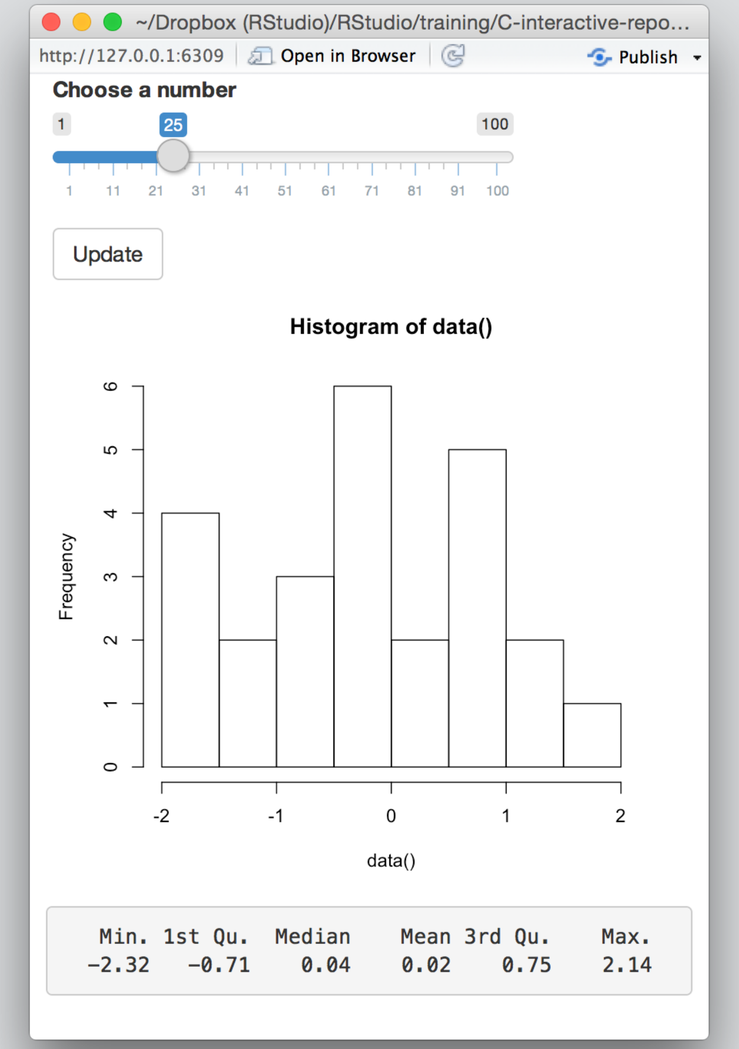
Notice: Id not ID

input name (for internal use)

label to display

```
actionButton(inputId = "go", label = "Click Me!")
```

The value of an action button increases by one each time it is pressed.

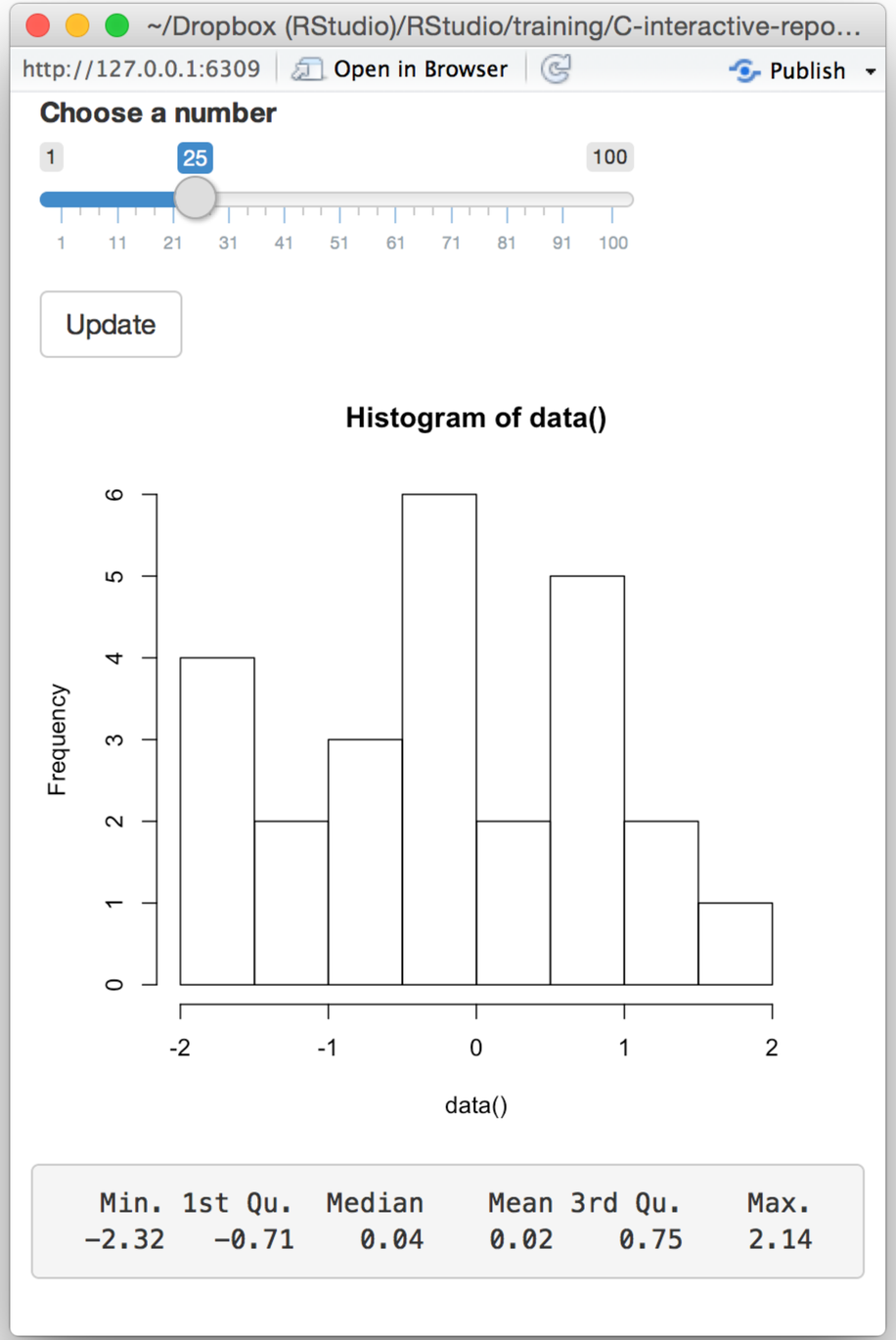


Your Turn

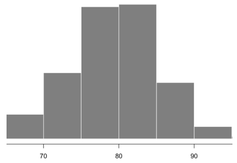
Add an **actionButton()** to the app. Then replace **reactive()** with **eventReactive()** so that the app only responds when the button is clicked.

Ensure that you can predict how the app will work.

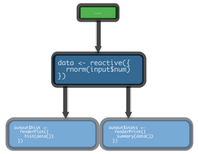
```
ui <- fluidPage(
  sliderInput("num", “Choose a number", 1, 100, 50),
  actionButton("go", "Update"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   data <- eventReactive(input$go, {rnorm(input$num)})
   output$hist <- renderPlot({
     hist(data())
   })
   output$sum <- renderPrint({
     summary(data())
   })
}
shinyApp(ui = ui, server = server)
```



Use…



render() to make an **object to display** in the UI.

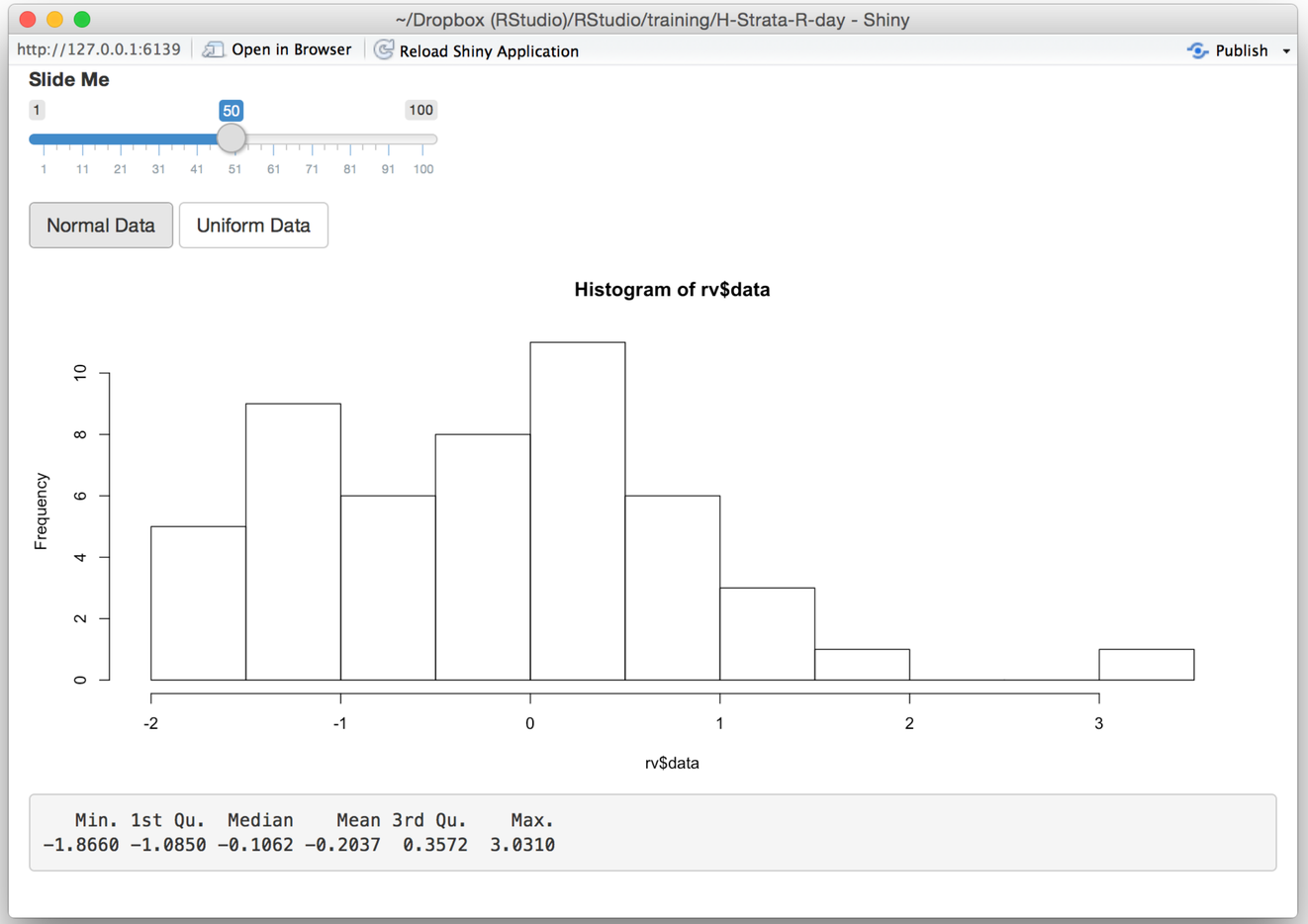


reactive() to make an **object to use** in downstream code.

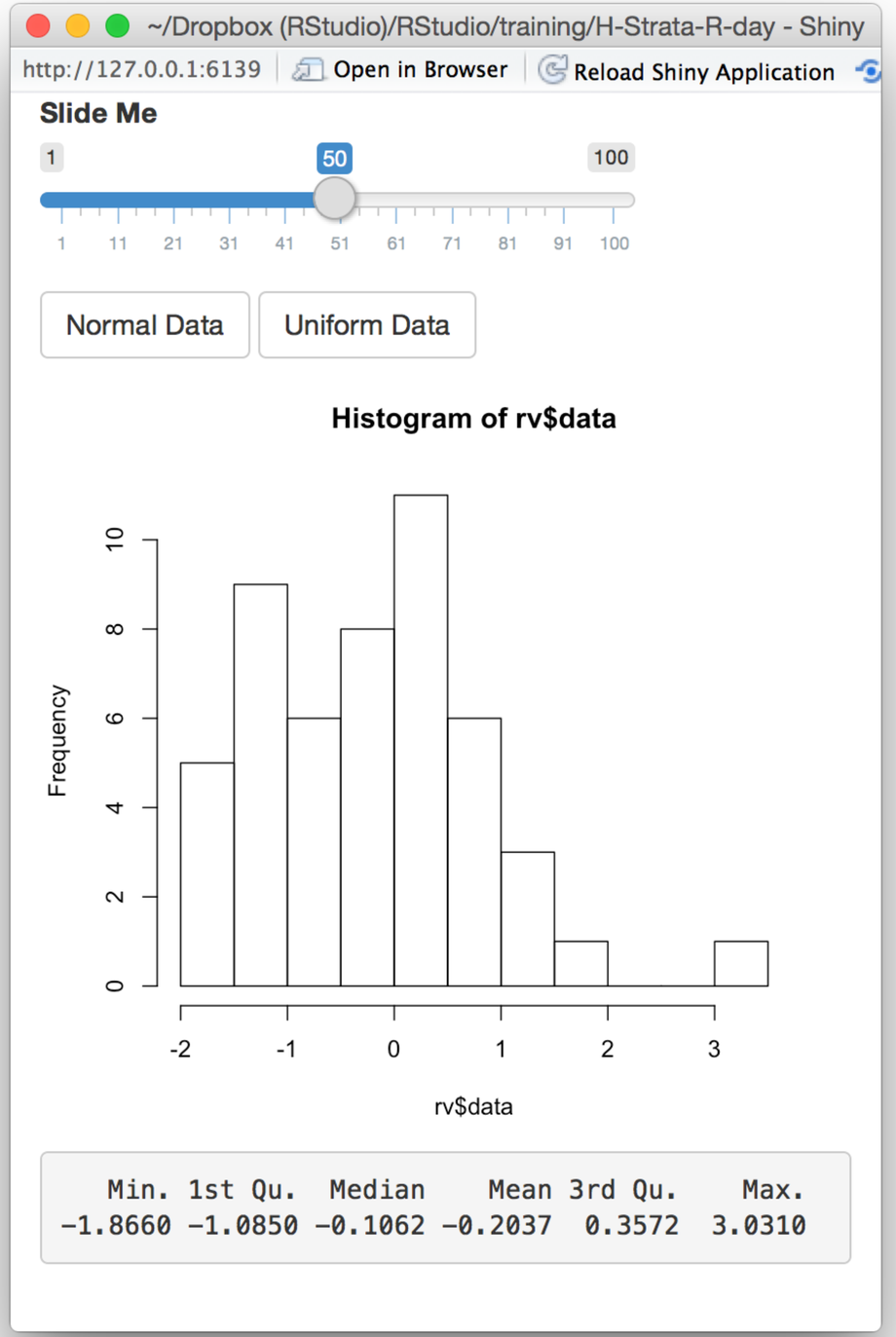
isolate() to return a **non-reactive object**.

eventReactive() to **delay a reaction**.

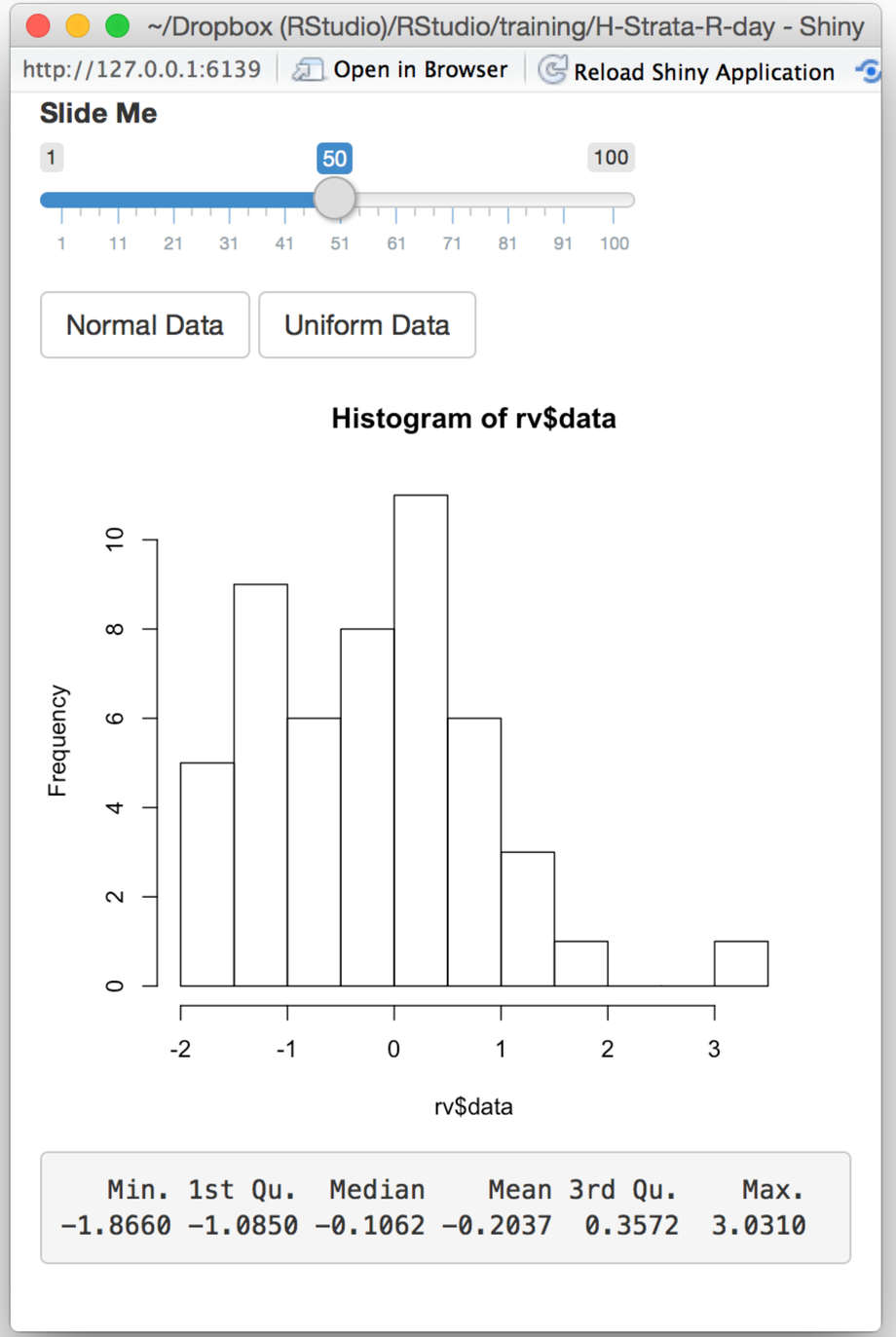
demo


~/Dropbox (RStudio)/RStudio/training/H-Strata-R-day - Shiny
http://127.0.0.1:6139
Open in Browser
Reload Shiny Application
Publish
Slide Me
1
50
100
1 11 21 31 41 51 61 71 81 91 100
Normal Data
Uniform Data
Histogram of rv$data
Frequency
0 2 4 6 8 10
-2 -1 0 1 2 3
rv$data
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.8660 -1.0850 -0.1062 -0.2037 0.3572 3.0310

```r
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```



```r
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```



observeEvent()

Triggers code to run.

```
observeEvent(input$norm, {rv$data <- rnorm(input$num)})
```

observeEvent()

Triggers code to run.

```
observeEvent(input$norm, {rv$data <- rnorm(input$num)})
```

runs the code block
(on the server side)

Builds an object that:

observeEvent()

Triggers code to run.

```
observeEvent(input$norm, {rv$data <- rnorm(input$num)})
```

Builds an object that:

runs the code block
(on the server side)

When notified by:

this or these reactive value(s)
and no others

input$norm

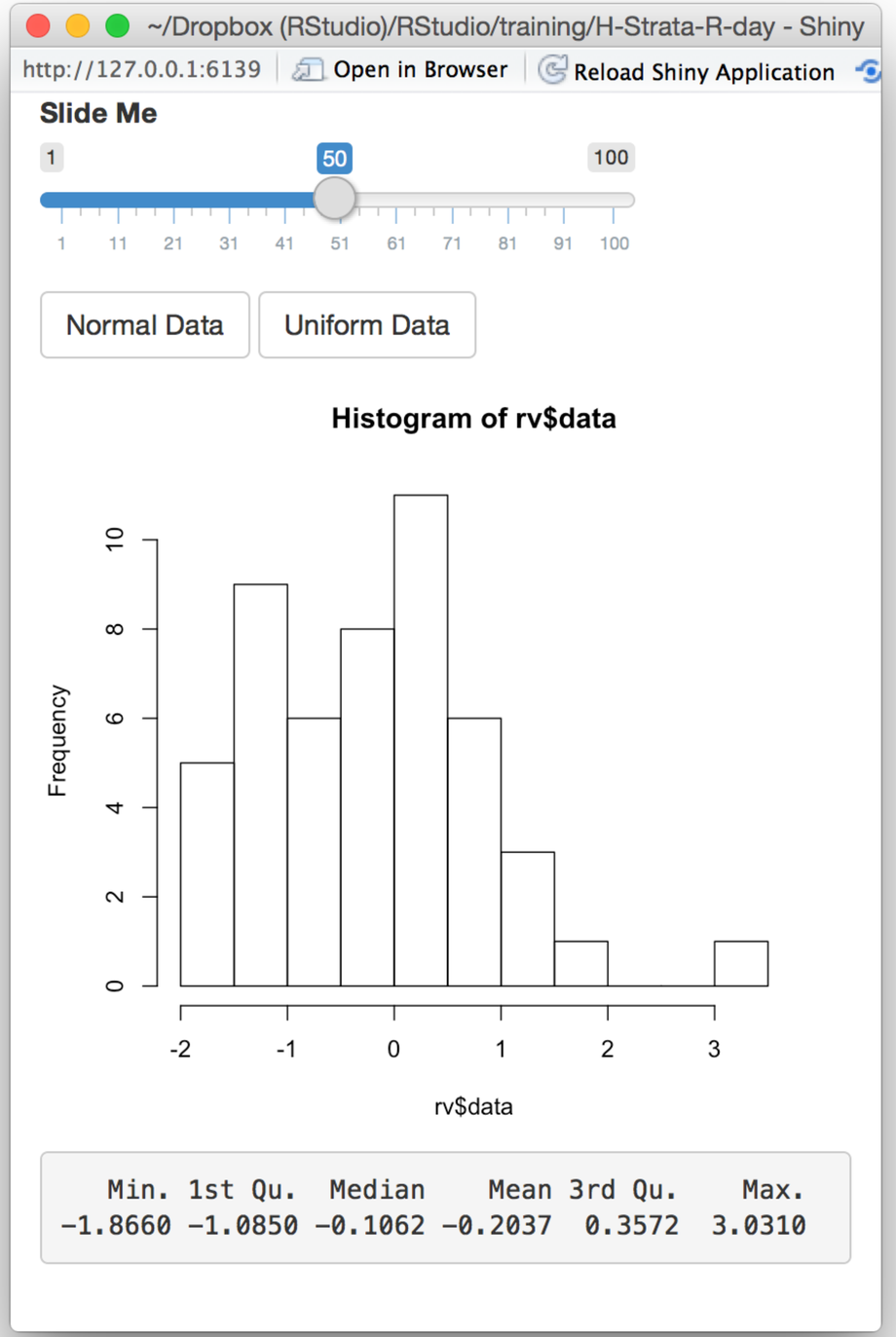
```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

input$unif

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

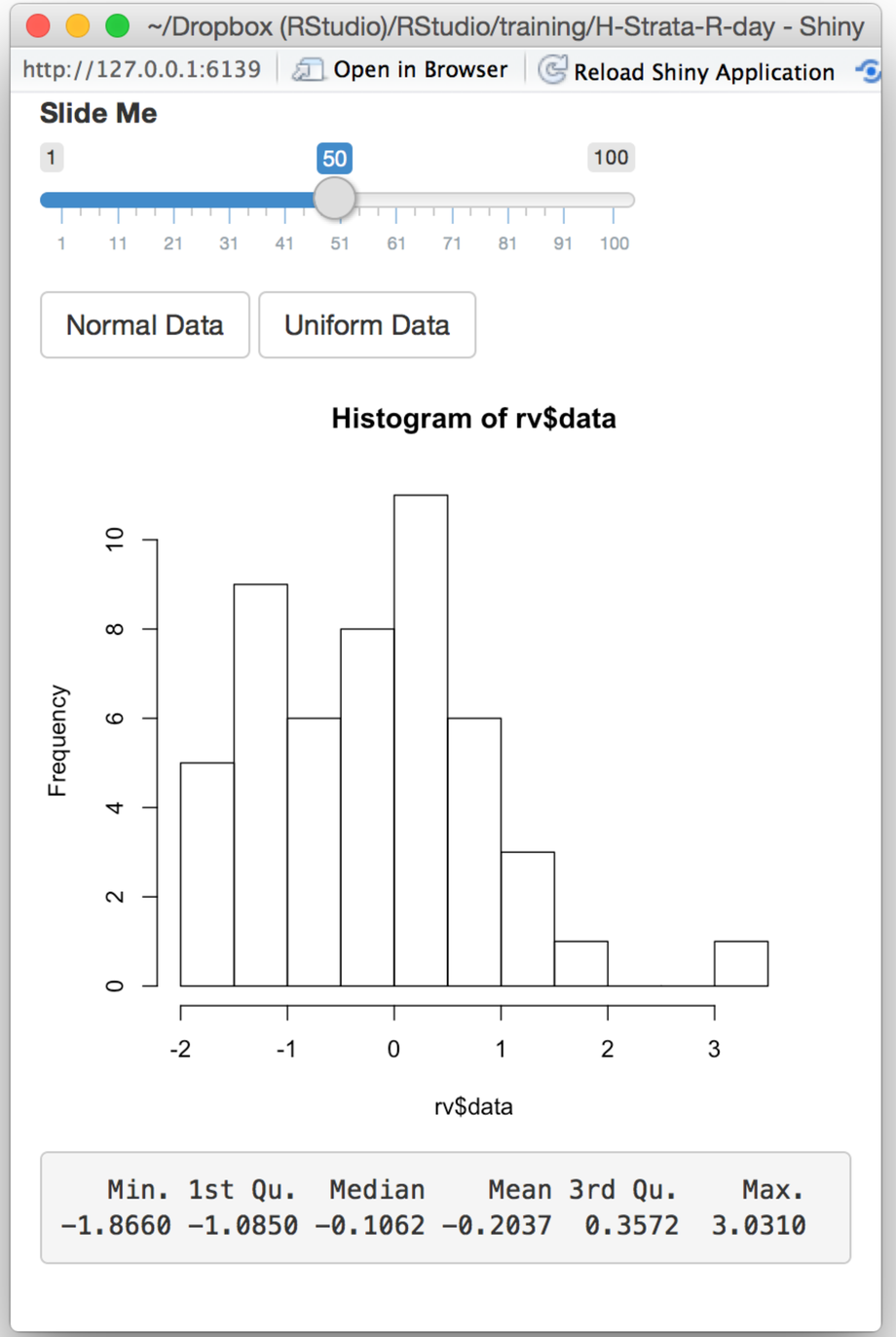
```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

input$unif

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
output$hist <-
   renderPlot({
     hist(data())
   })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

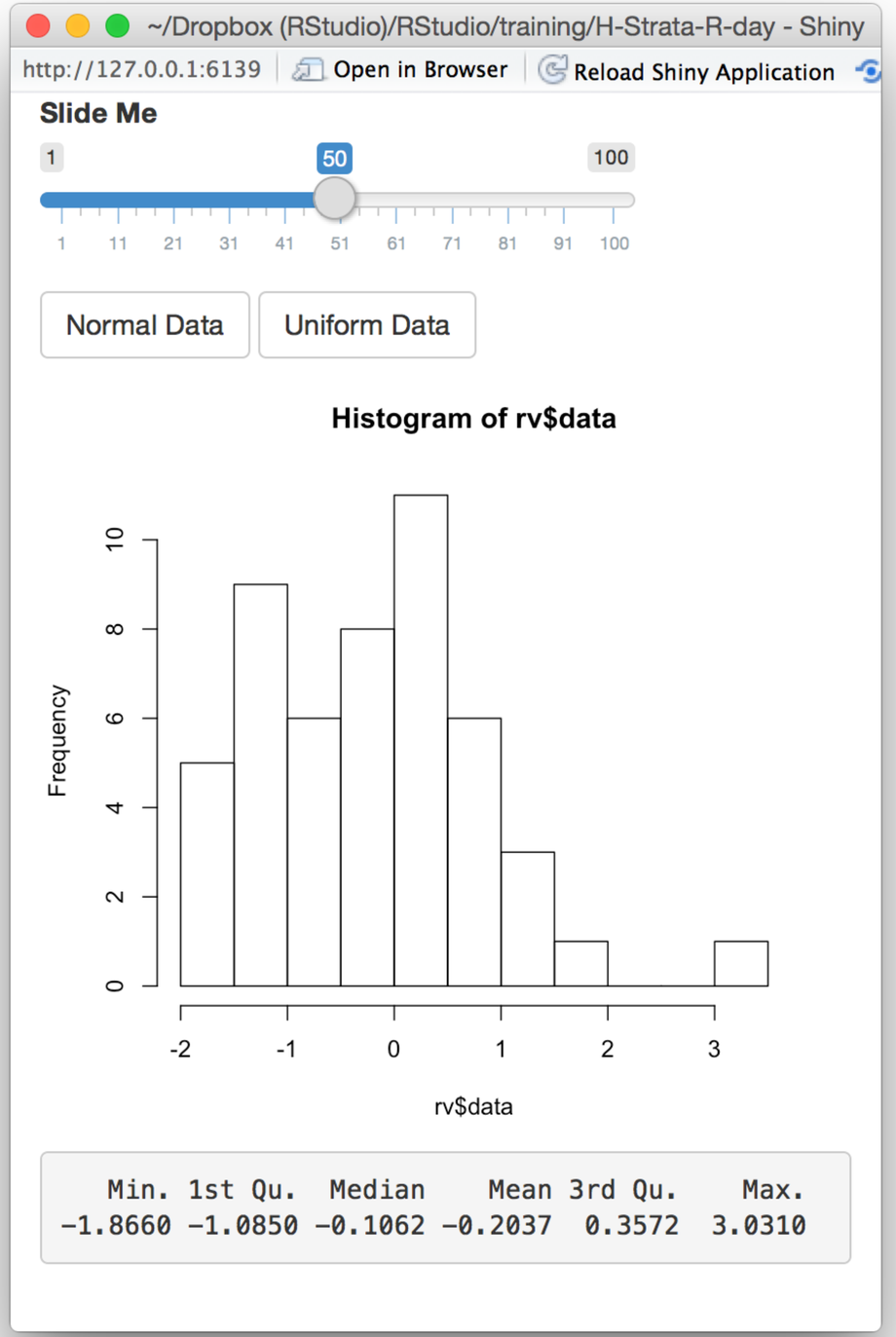
```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
   })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

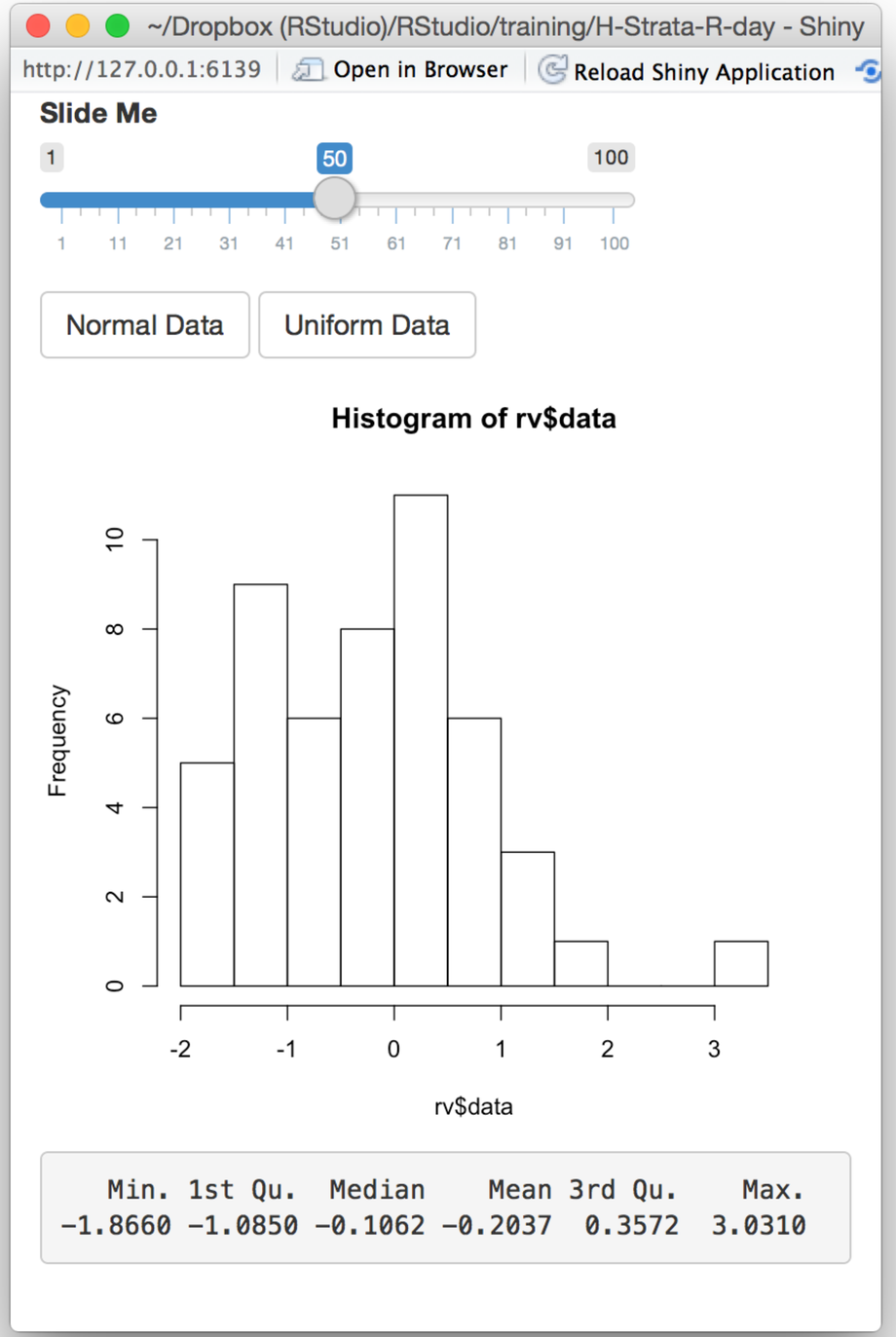
```
output$hist <-
   renderPlot({
     hist(data())
  })
```

input$unif

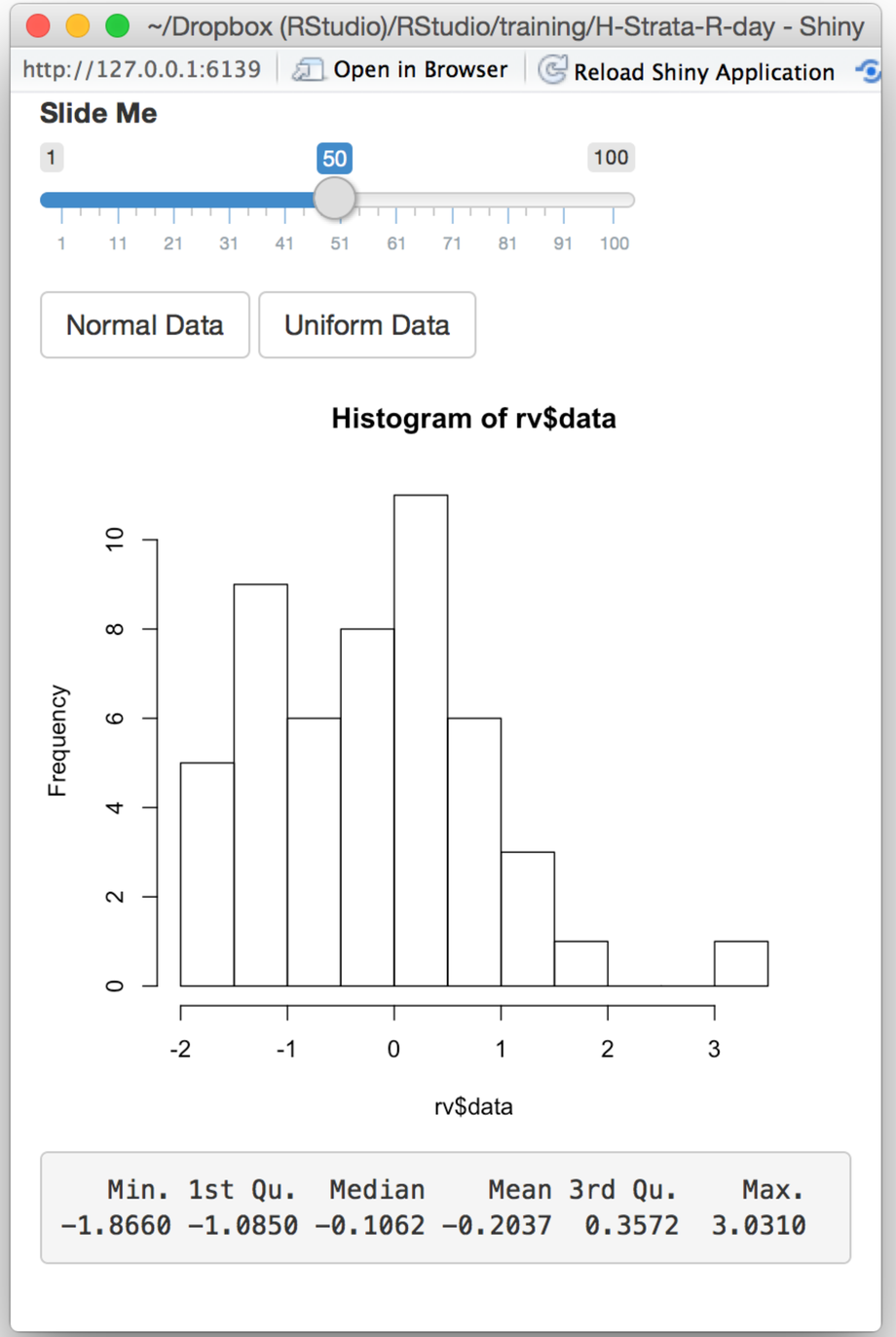
```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ runif(input$num)
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```

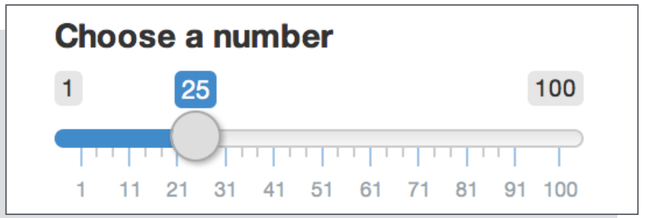


```r
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```

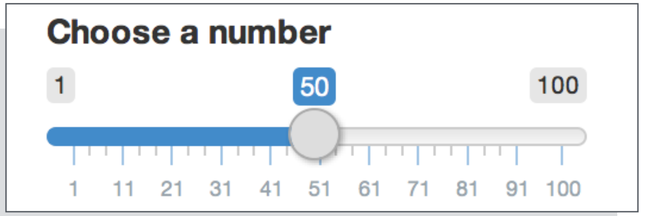


Reactive values

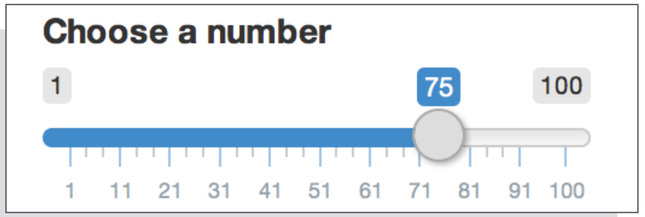
The input list contains values that change whenever a user changes an input.




```
input$num = 25
```



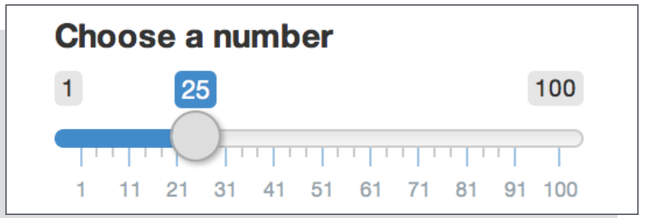

```
input$num = 50
```



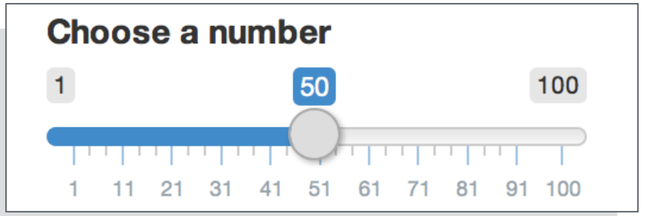

```
input$num = 75
```

Reactive values

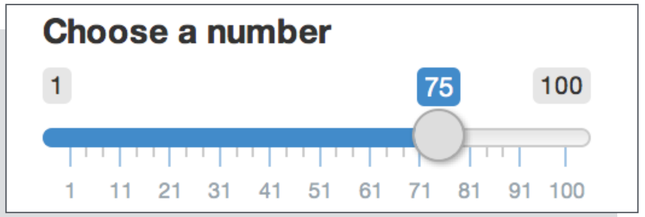
The input list contains values that change whenever a user changes an input.




```
input$num = 25
```




```
input$num = 50
```




```
input$num = 75
```

You **cannot** set these values in your code

reactiveValues()

Creates a list of reactive values that you can manipulate

```
rv <- reactiveValues(data = rnorm(100))
```

reactiveValues()

Creates a list of reactive values that you can manipulate

```
rv <- reactiveValues(data = rnorm(100))
```

(optional) elements to add to the list

reactiveValues()

Creates a list of reactive values that you can manipulate

```
rv <- reactiveValues(data = rnorm(100))
```

(optional) elements to add to the list

Builds a list of objects that:

notify objects that use them that the objects are invalid

reactiveValues()

Creates a list of reactive values that you can manipulate

```
rv <- reactiveValues(data = rnorm(100))
```

(optional) elements to add to the list

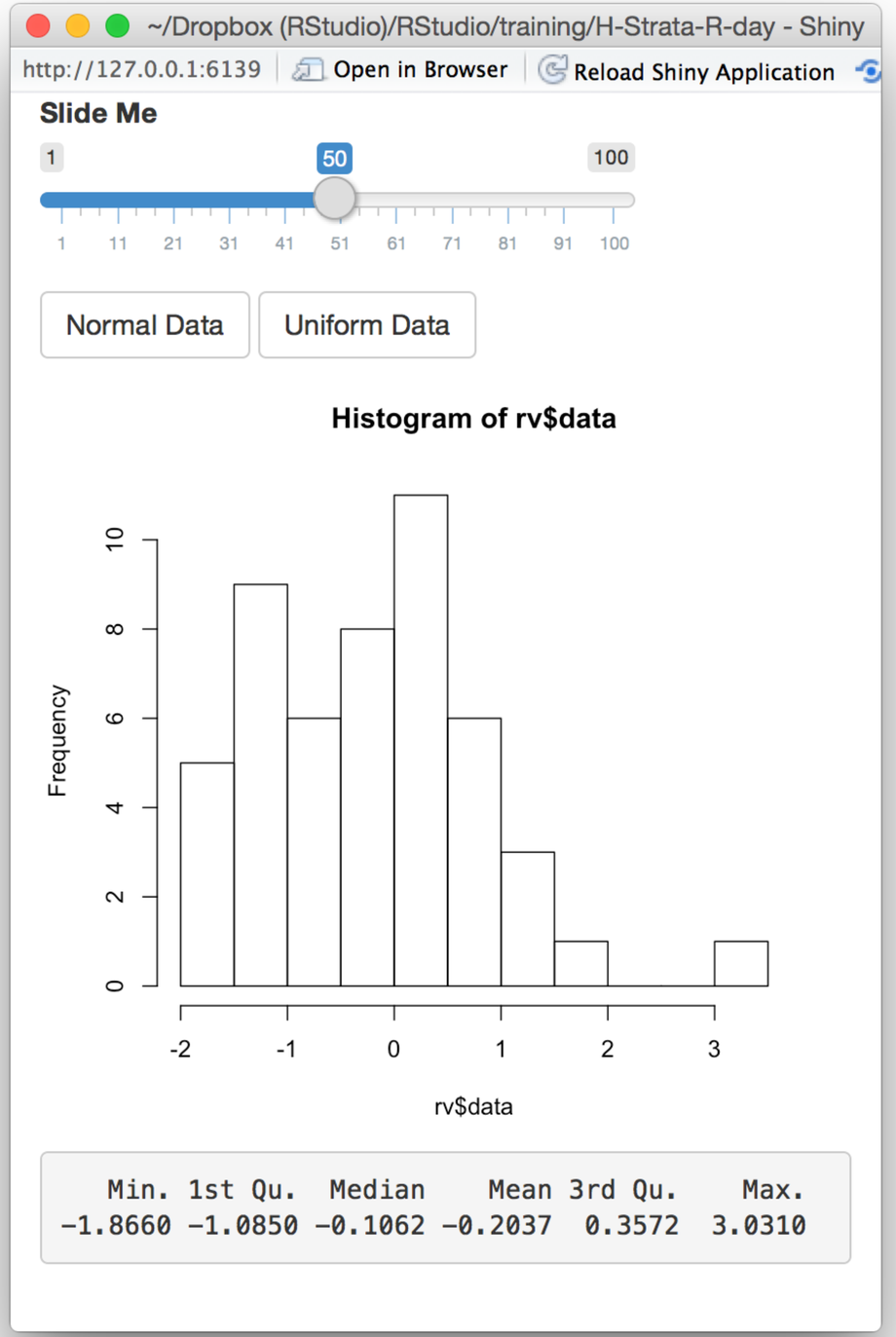
Builds a list of objects that:

notify objects that use them that the objects are invalid

When:

When their own value changes

```r
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```



input$norm

input$unif

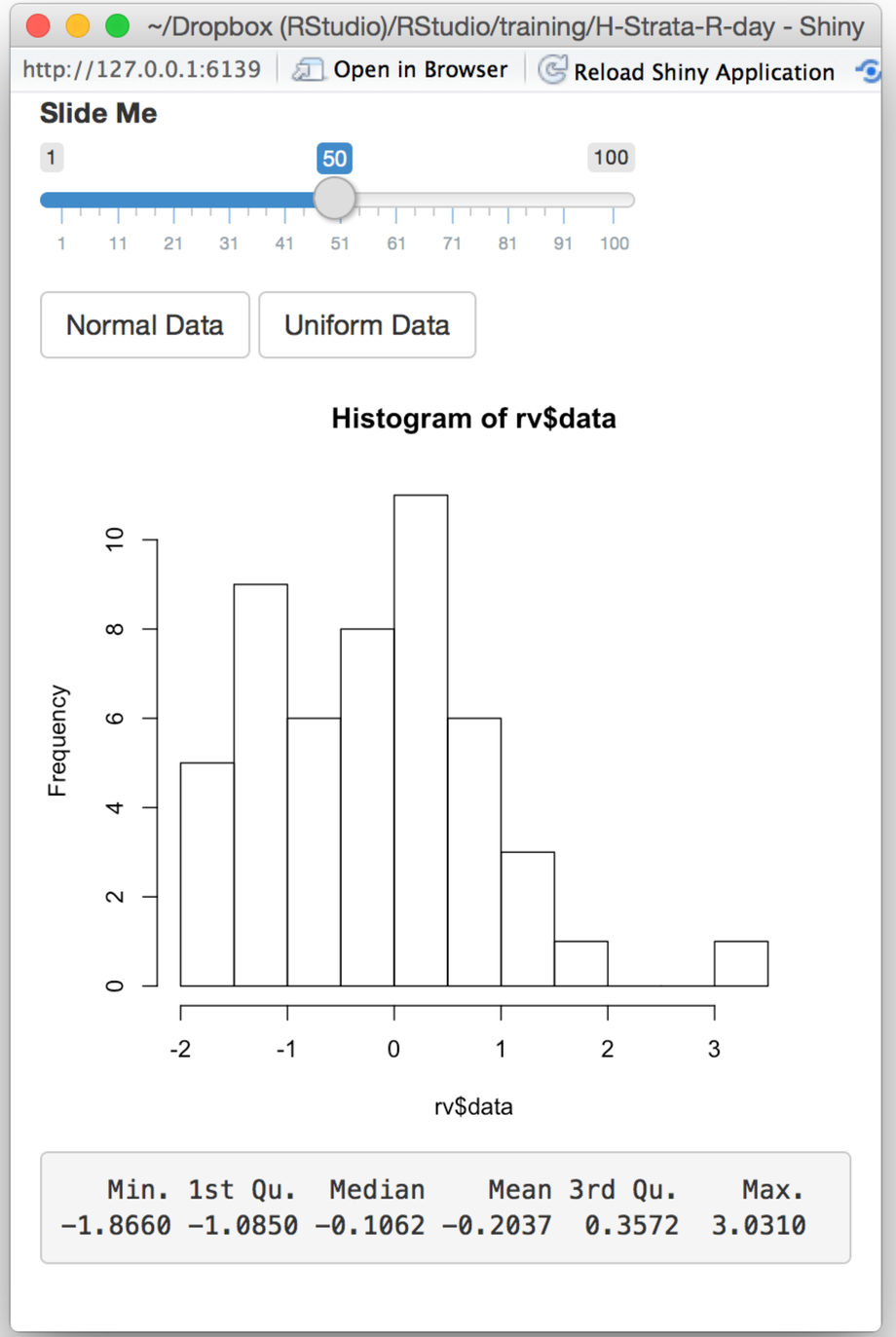
```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

rv$data
rnorm(50)

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

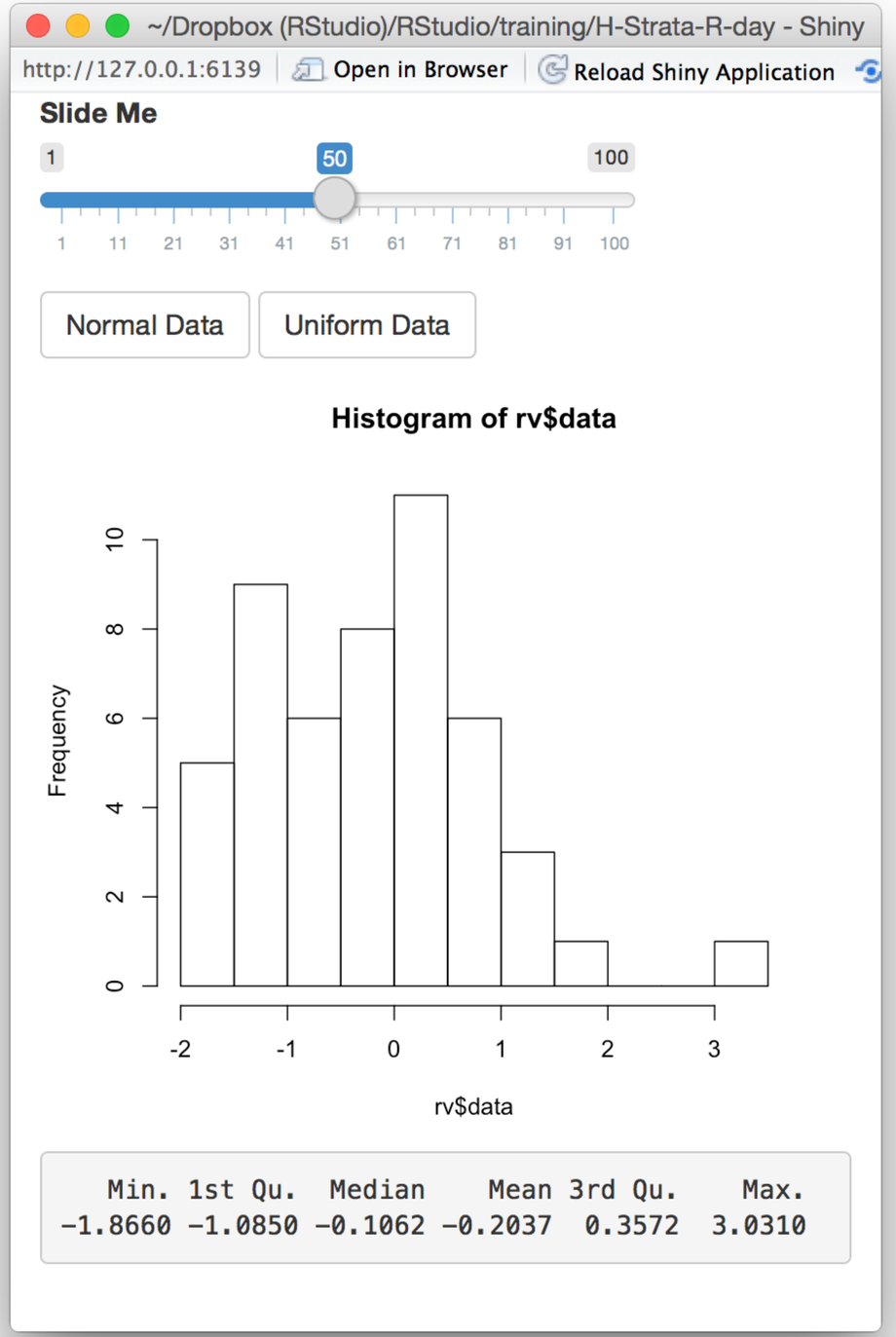
rv$data
rnorm(50)

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

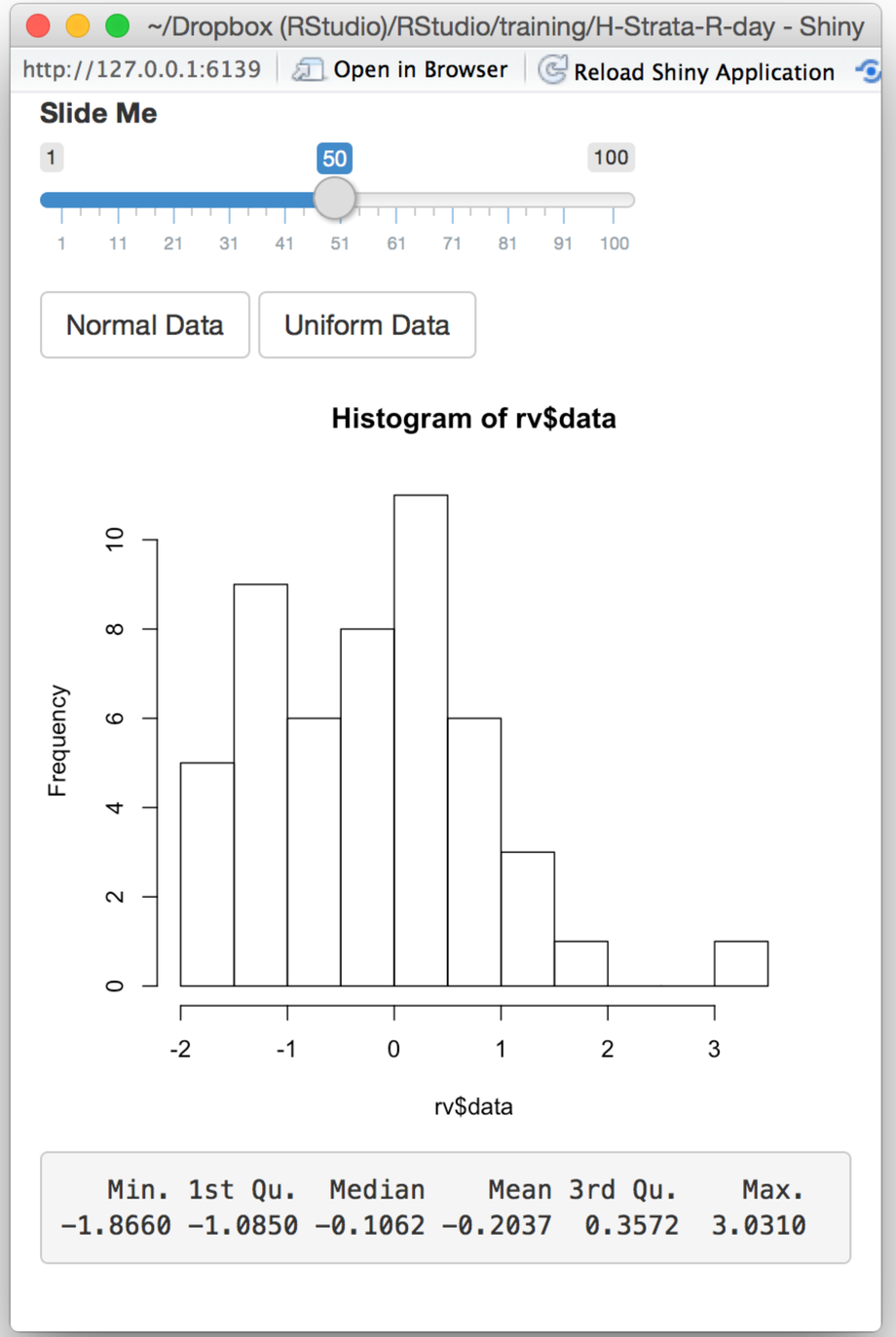
```
rv$data
rnorm(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

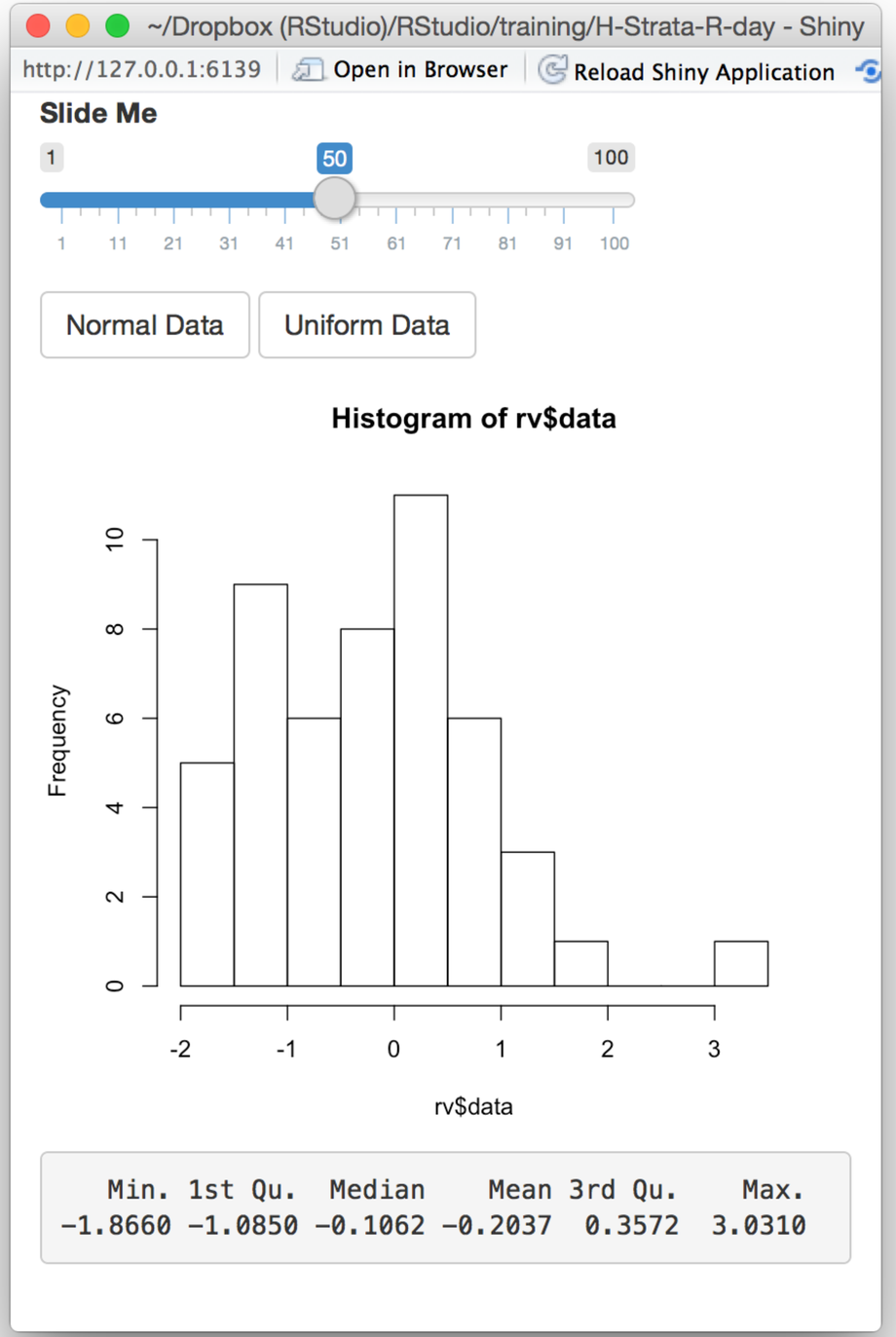
```
rv$data
rnorm(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

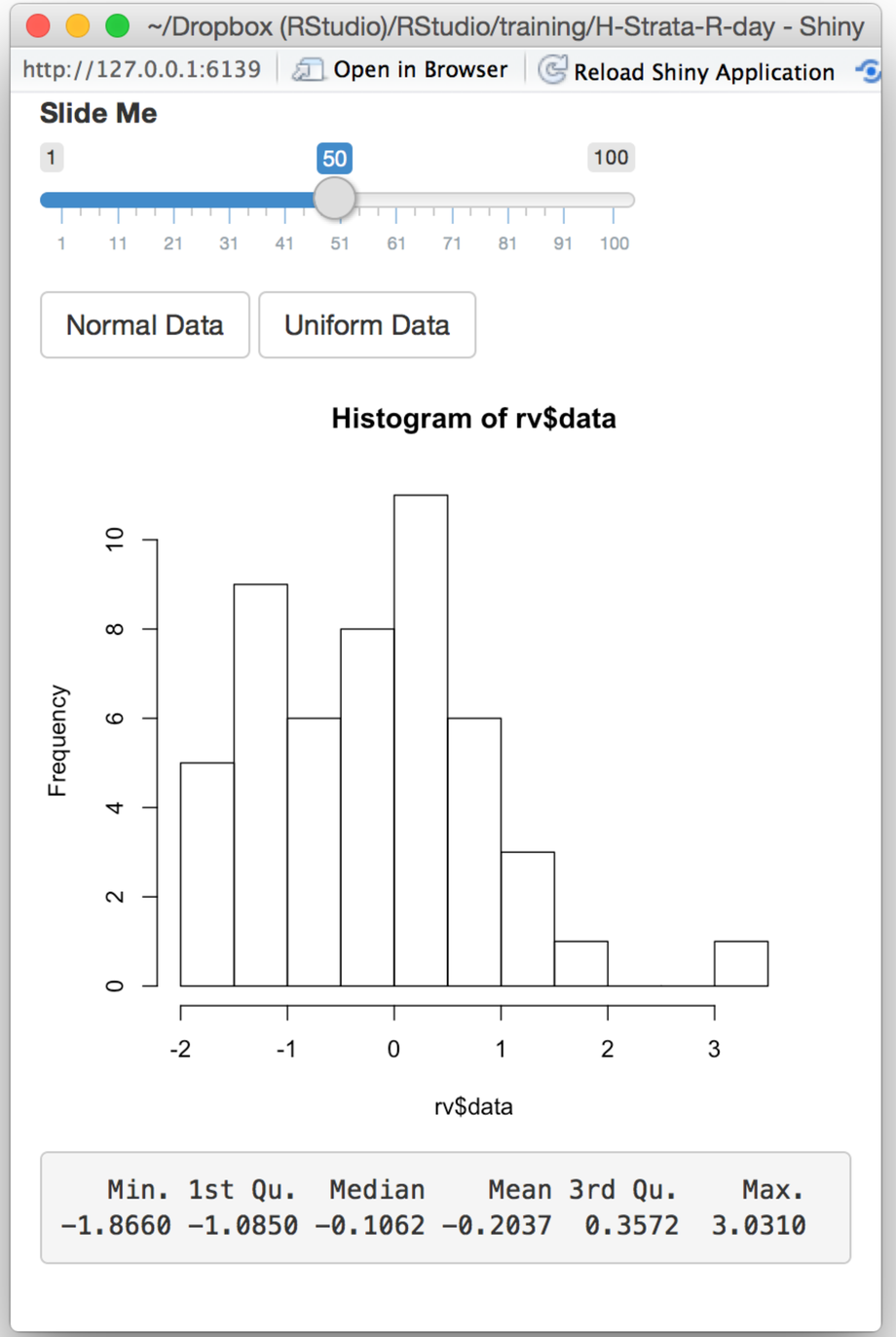
```
rv$data
rnorm(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

```
rv$data
rnorm(
input$num)
```

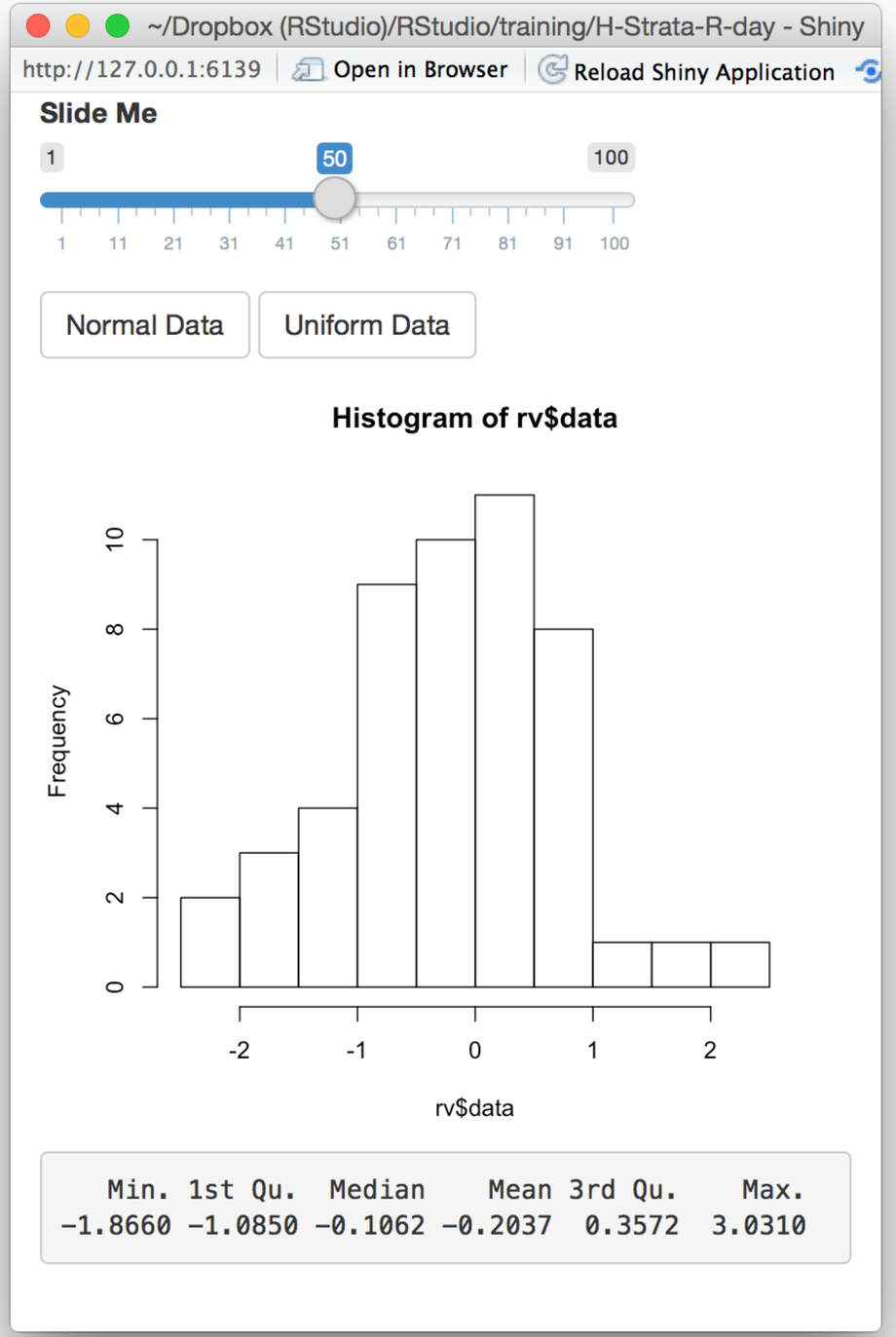
```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

rv$data
rnorm(
input$num)

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

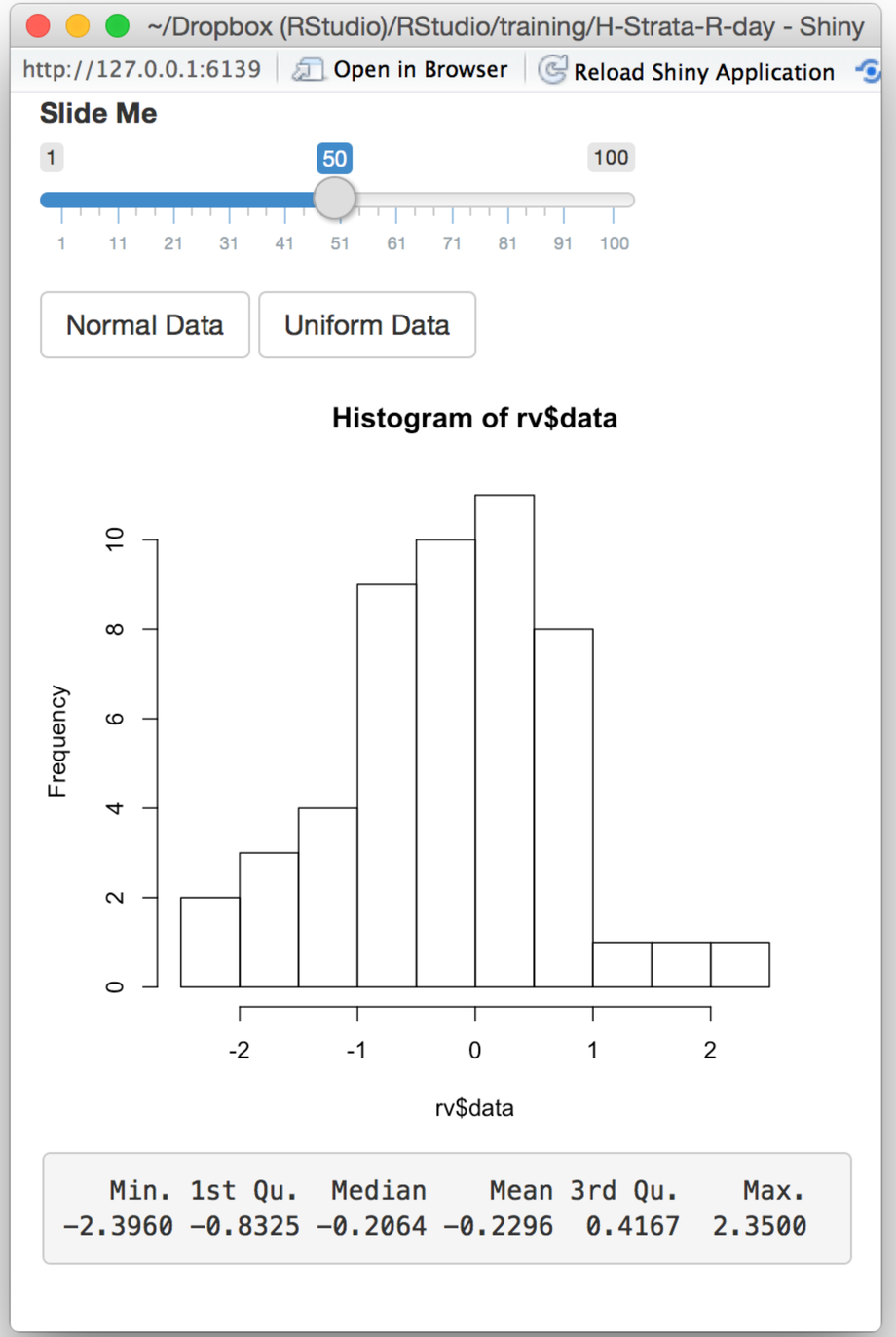
```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```

```
> summary(data())
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

rv$data
rnorm(
input$num)

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

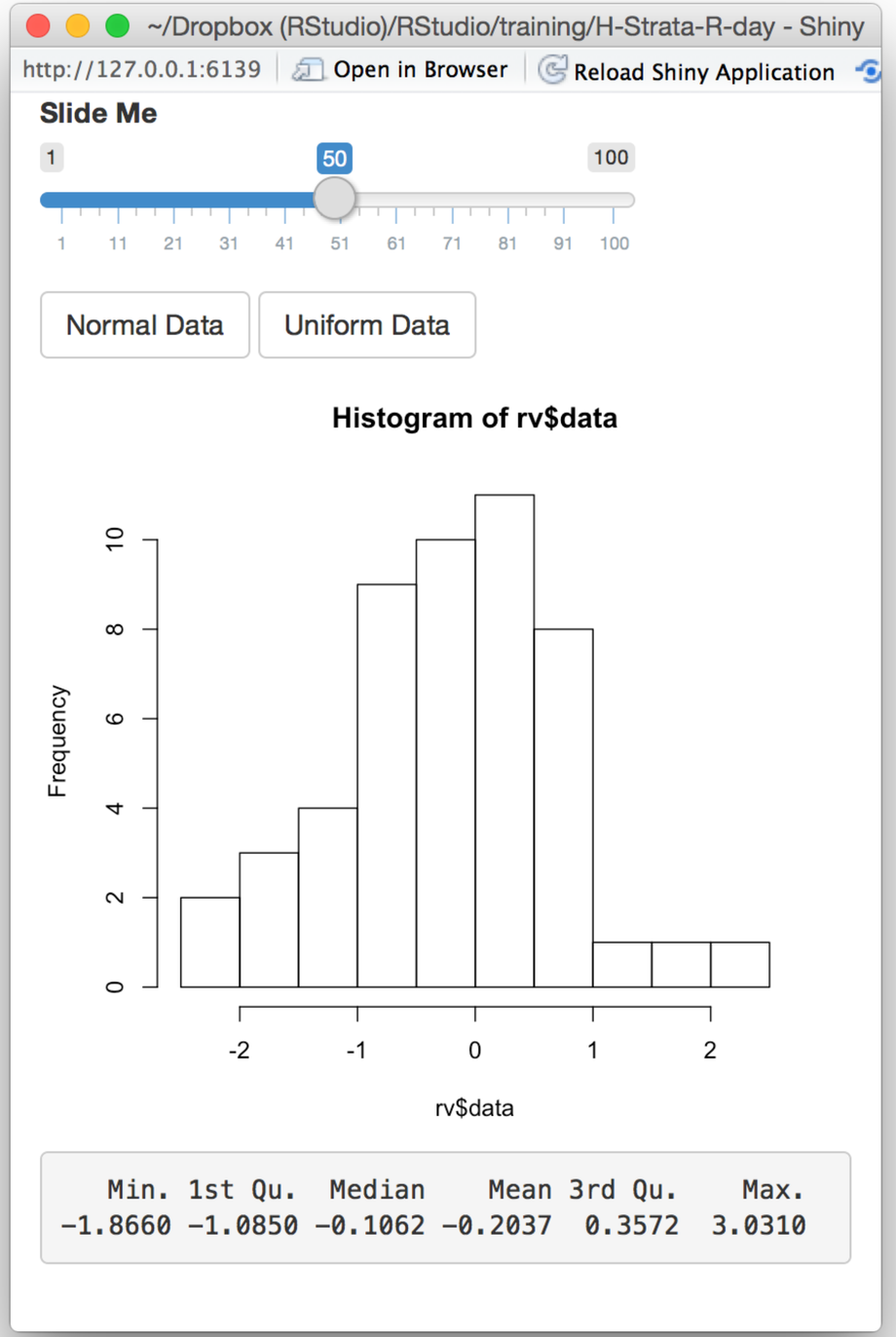
```
> rv$data <-
+ rnorm(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```

```
> summary(data())
```



input$norm

input$unif

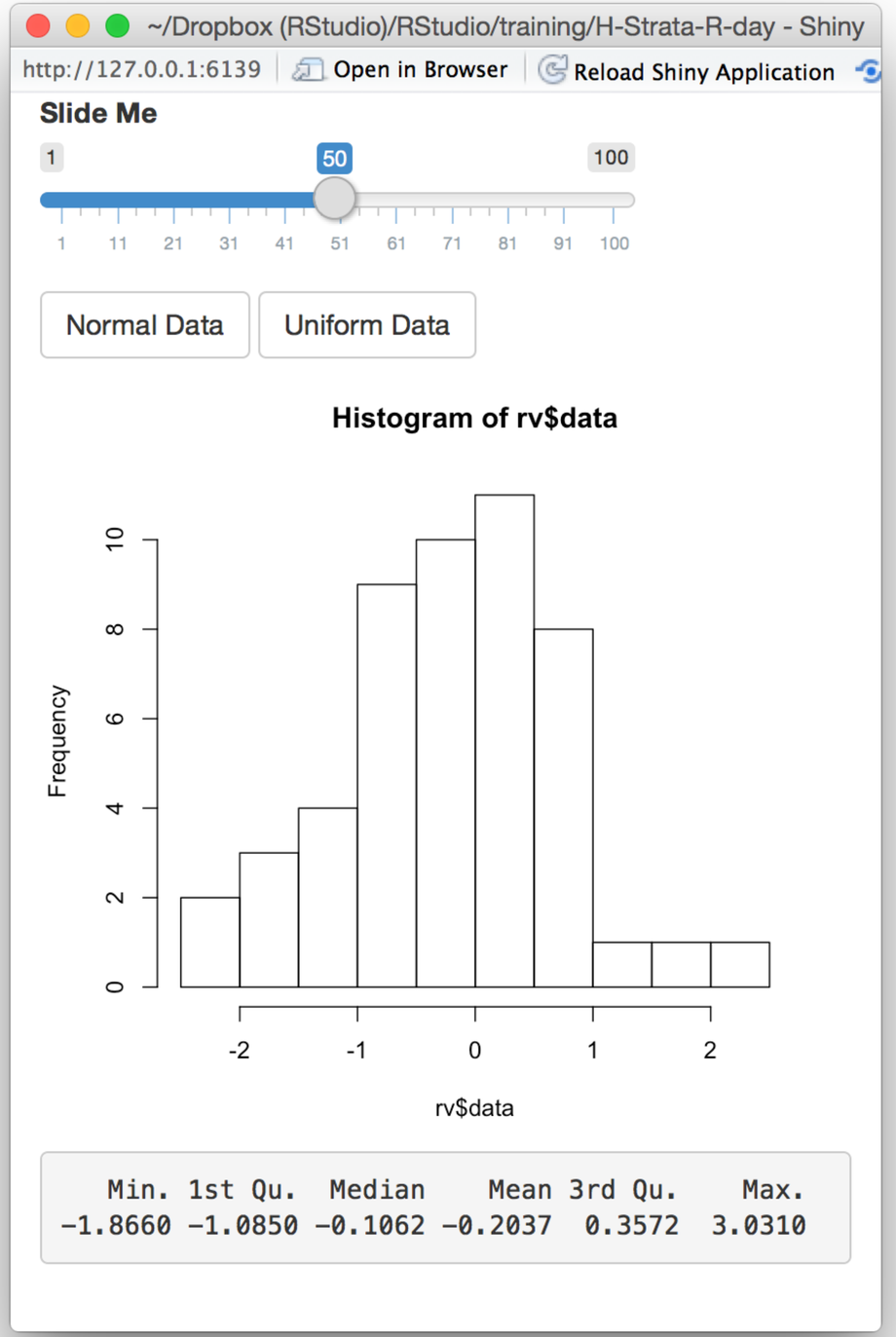
```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

rv$data
rnorm(
input$num)

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

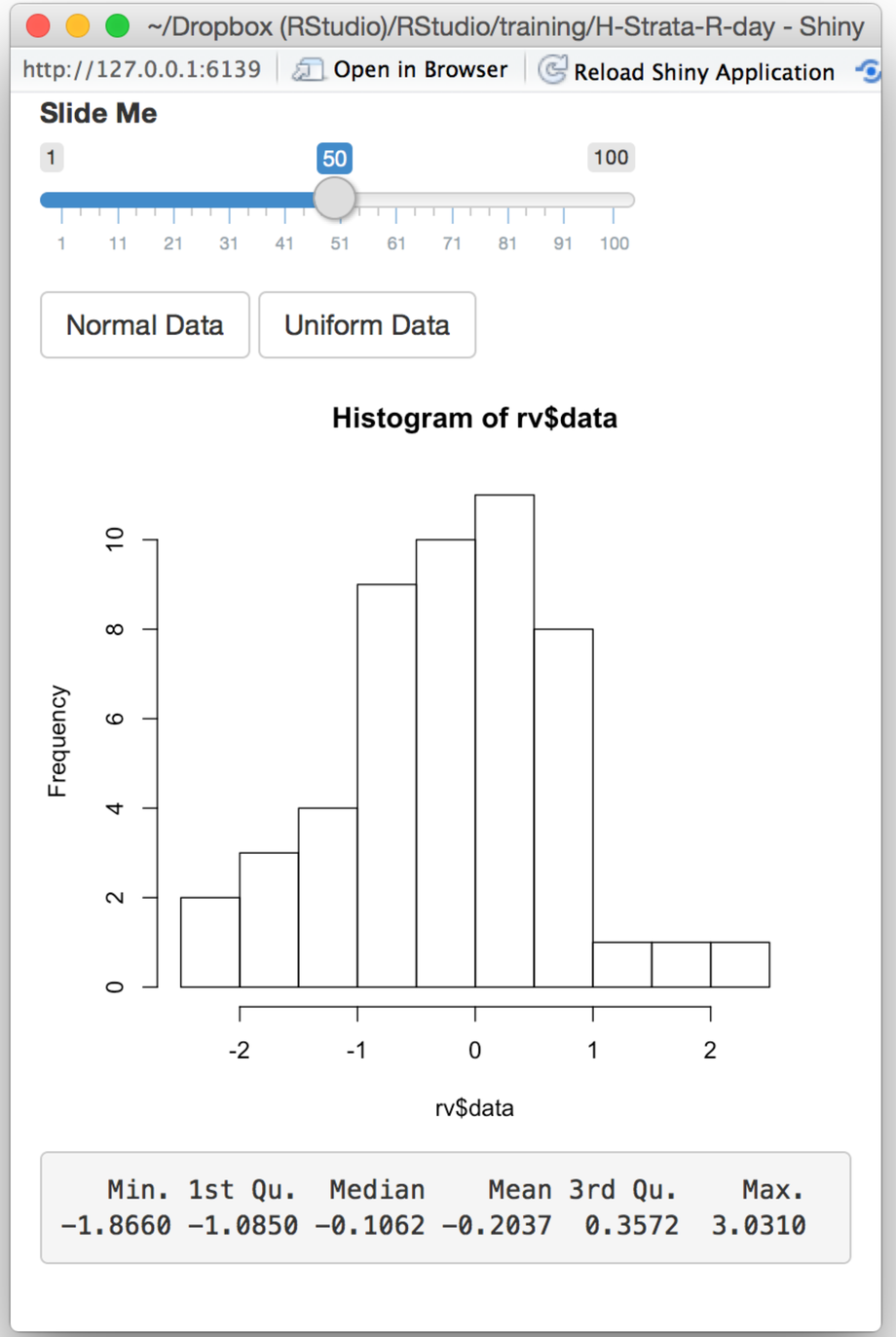
```
rv$data
rnorm(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ runif(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

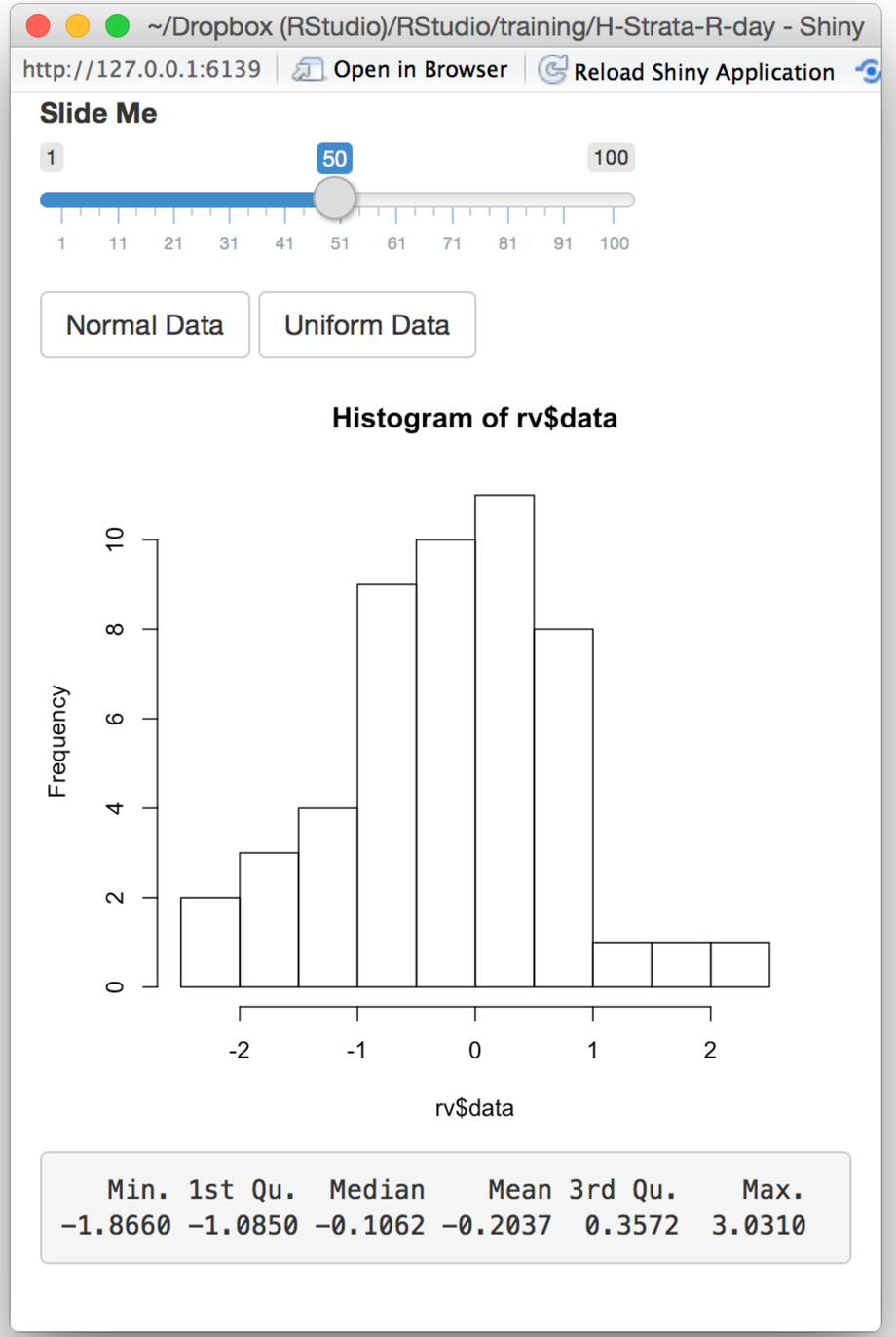
```
rv$data
runif(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ runif(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

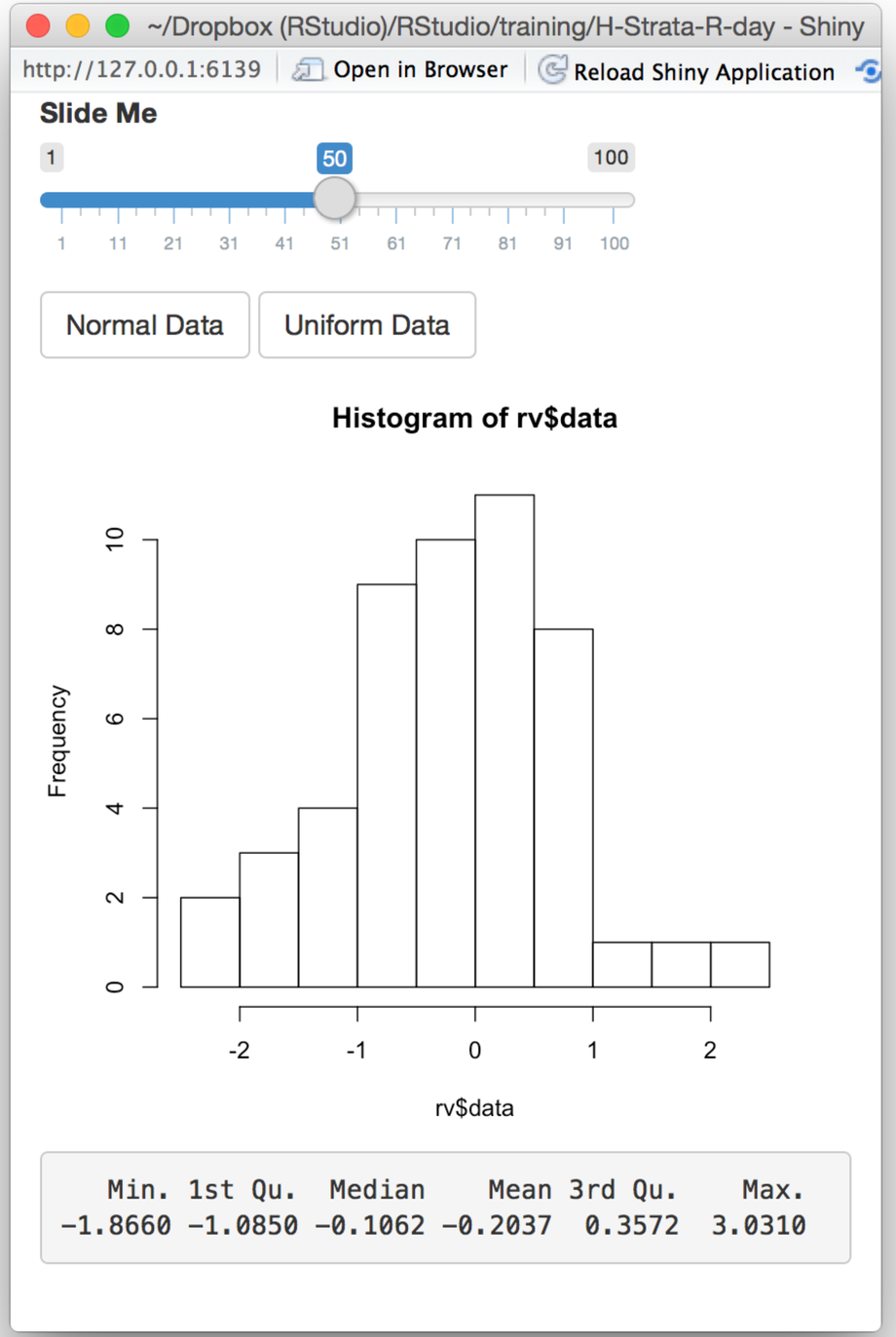
```
rv$data
runif(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ runif(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

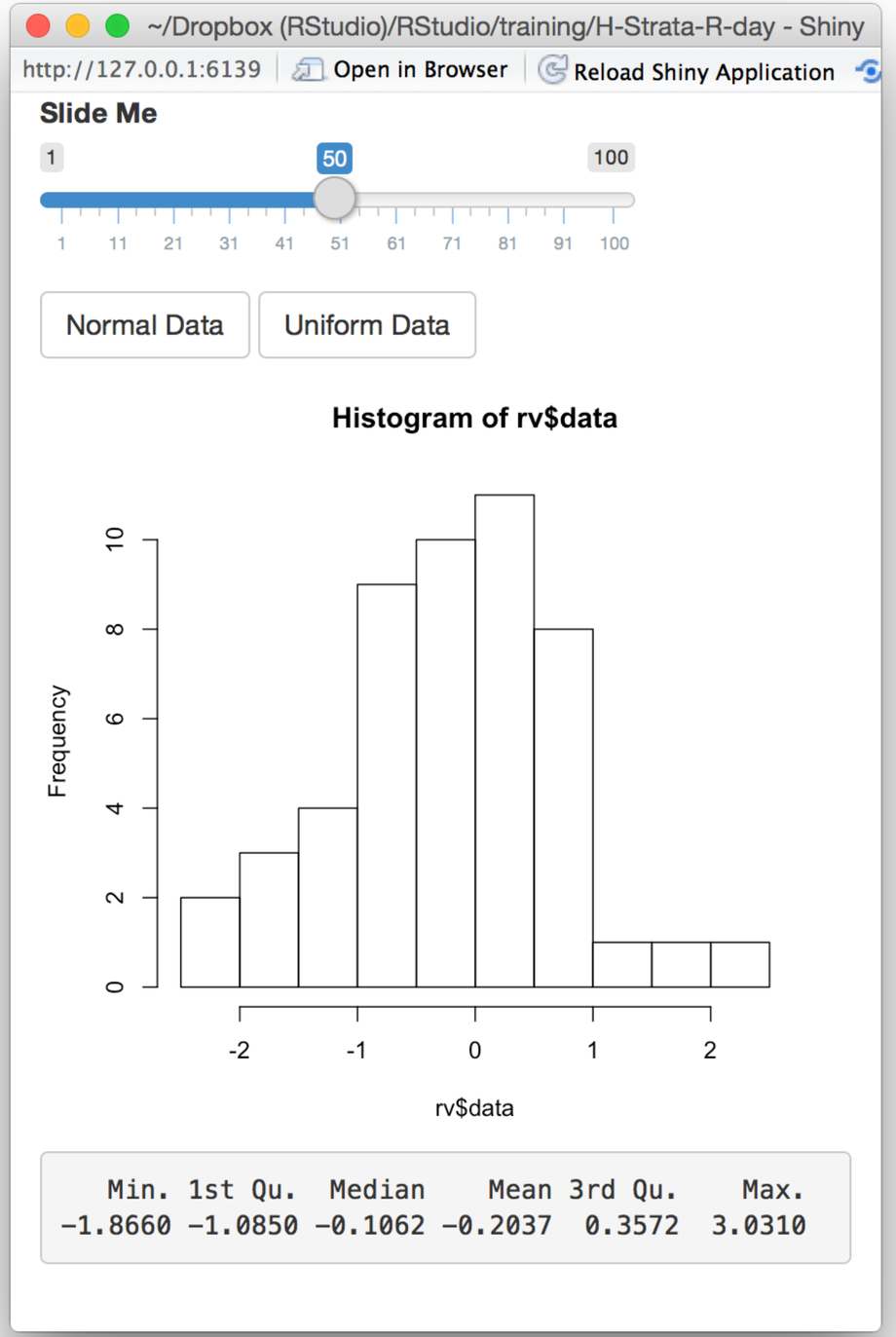
rv$data
runif(
input$num)

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ runif(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

rv$data
runif(
input$num)

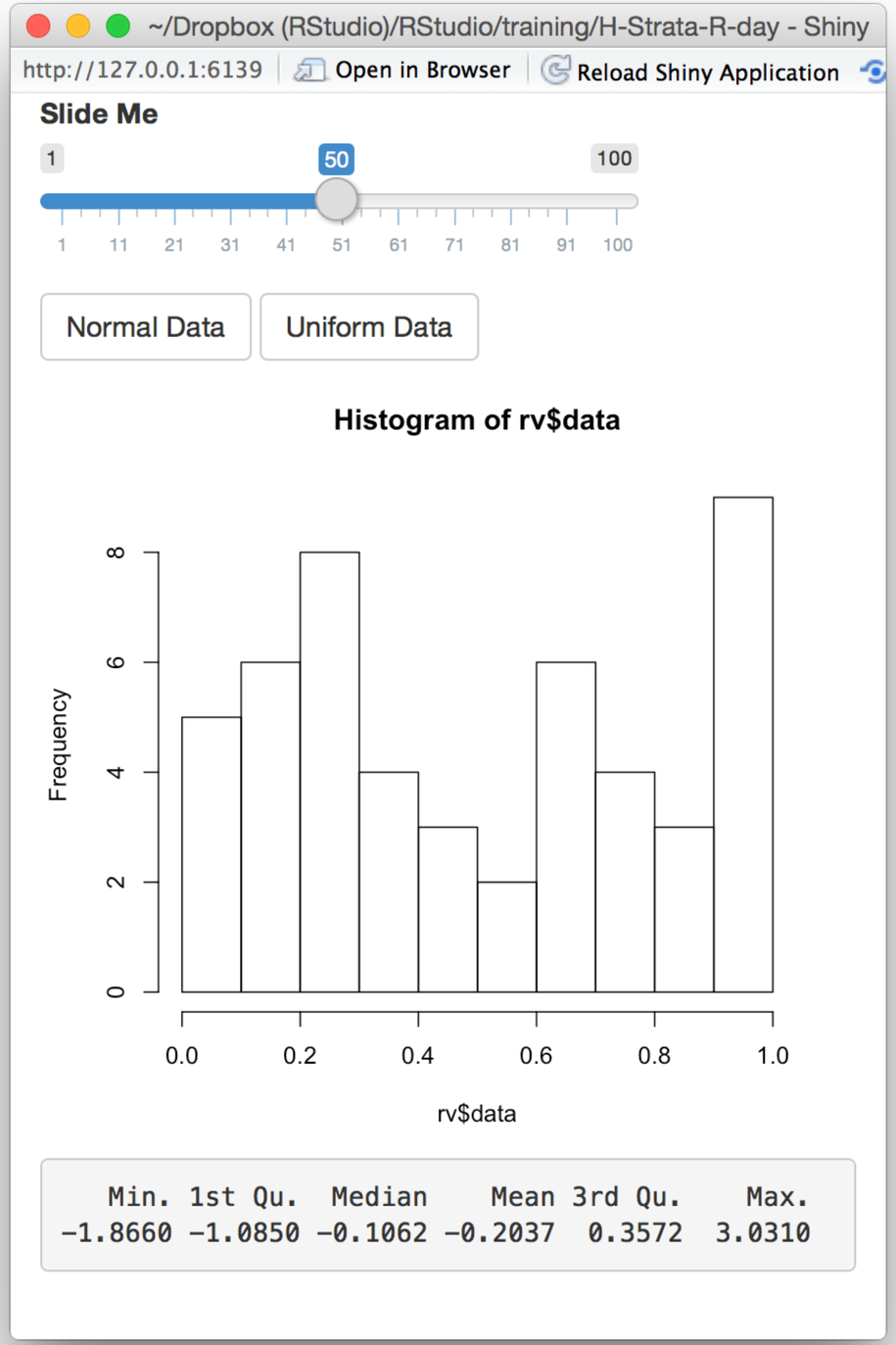
```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ runif(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```



input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

rv$data
runif(
input$num)

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

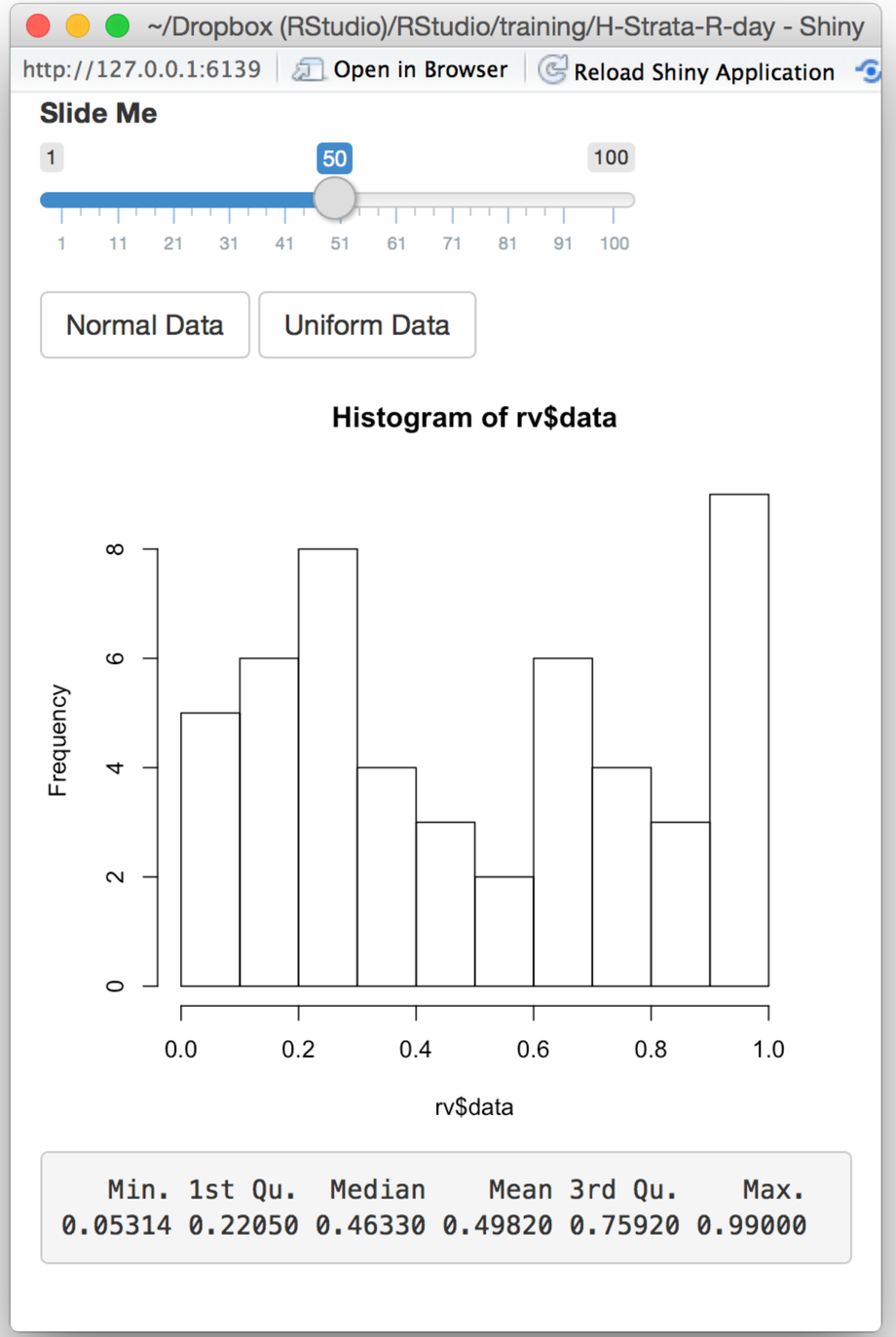
```
> rv$data <-
+ runif(input$num)
```

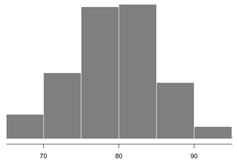
```
output$hist <-
   renderPlot({
     hist(data())
  })
```

```
output$stats <-
   renderPrint({
    summary(data())
  })
```

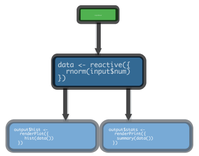
```
> hist(data()))
```

```
> summary(data())
```





render() to make an **object to display** in the UI.



reactive() to make an **object to use** in downstream code.

isolate() to return a **non-reactive object**.

eventReactive() to **delay a reaction**.

observeEvent() to **trigger code**.

```
rv$data <-
```

reactiveValues() to **make your own** reactive values.

observe()

Also triggers code to run on server.

Uses same syntax as render*(), reactive(), and isolate()

```
observe({rv$data <- rnorm(input$num)})
```

observe()

Also triggers code to run on server.

Uses same syntax as render*(), reactive(), and isolate()

```
observe({rv$data <- rnorm(input$num)})
```

runs the code block
(on the server side)

Builds an object that:

observe()

Also triggers code to run on server.

Uses same syntax as render*(), reactive(), and isolate()

```
observe({rv$data <- rnorm(input$num)})
```

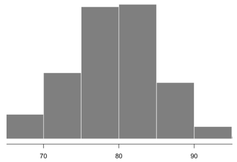
Builds an object that:

runs the code block
(on the server side)

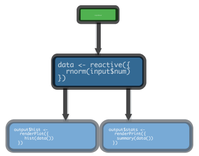
When notified by:

any reactive value in the code block

Use…



render() to make an **object to display** in the UI.



reactive() to make an **object to use** in downstream code.

isolate() to return a **non-reactive object**.

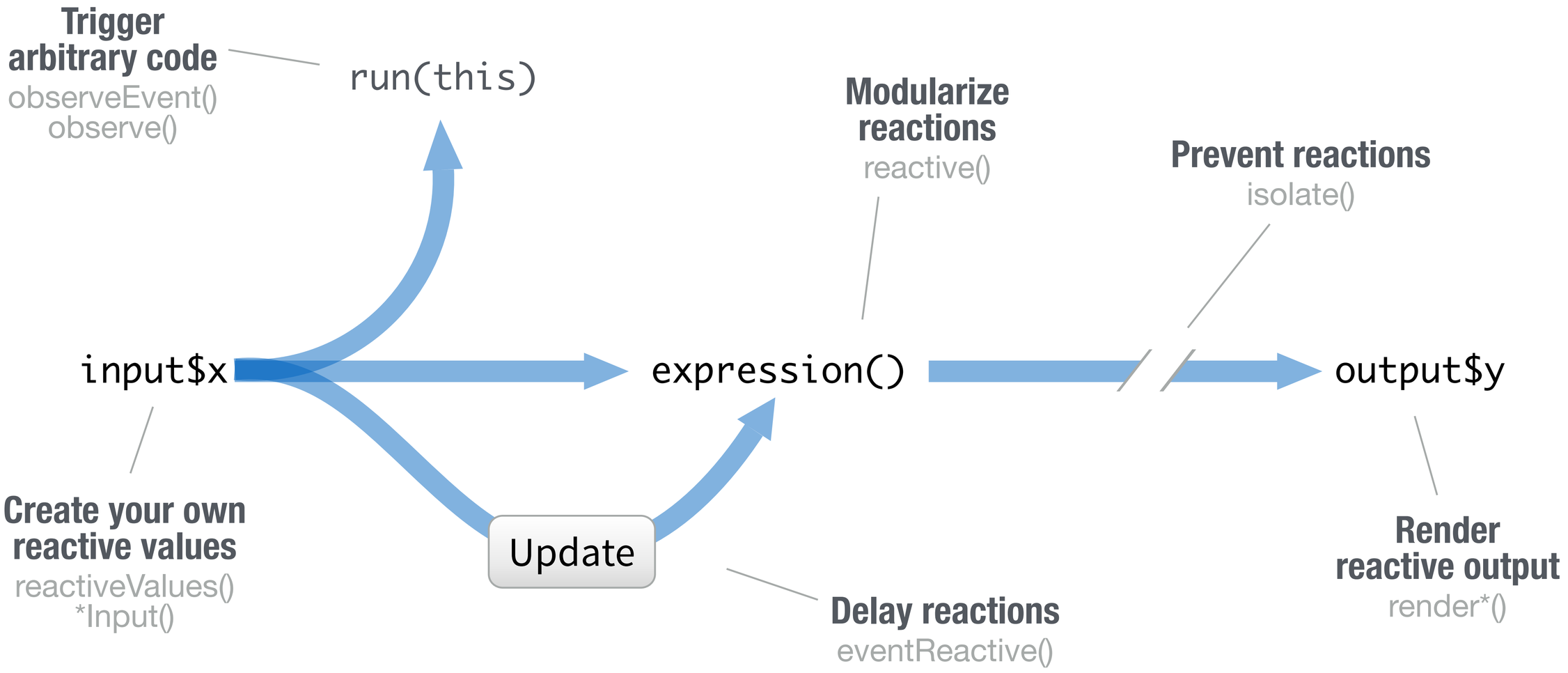
eventReactive() to **delay a reaction**.

observeEvent() or **observe()** to **trigger code**.

```
rv$data <-
```

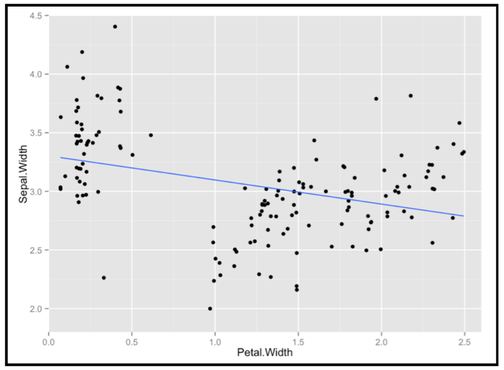
reactiveValues() to **make your own** reactive values.

Recap

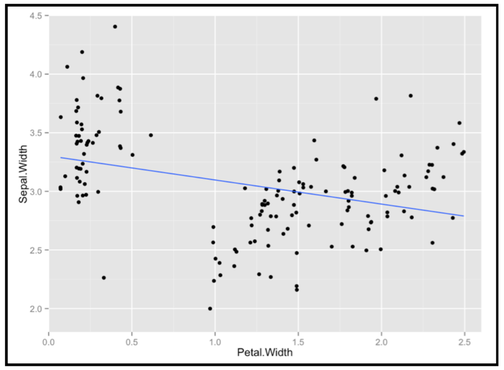

Trigger arbitrary code
observeEvent()
observe()
run(this)
Modularize reactions
reactive()
Prevent reactions
isolate()
input$x
expression()
output$y
Create your own reactive values
reactiveValues()
*Input()
Update
Delay reactions
eventReactive()
Render reactive output
render*()

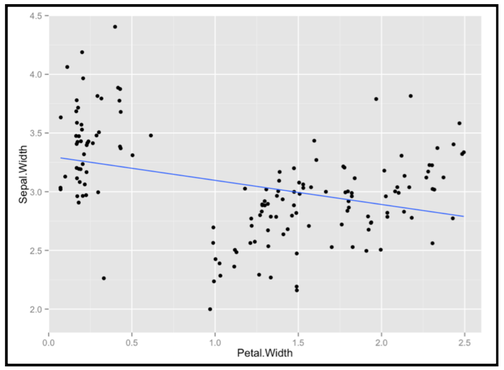
Interactive plots

Plots and images can be both outputs



Plots and images can be both outputs *and* inputs.

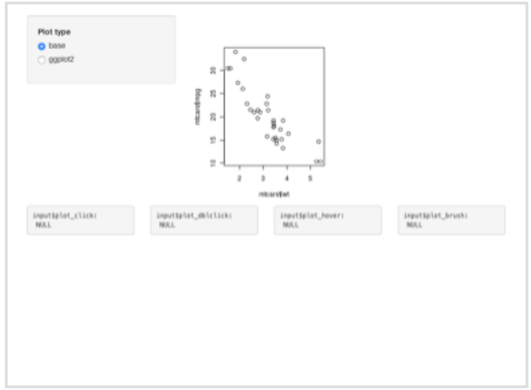




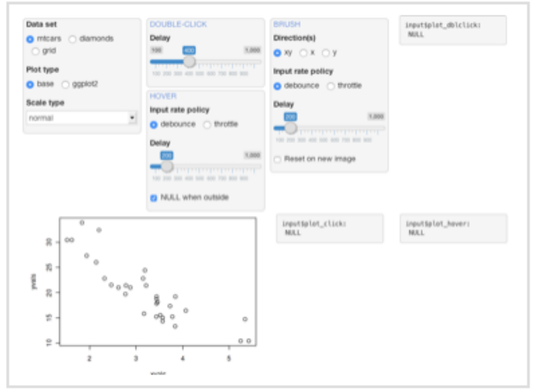
demos from the **Shiny Gallery**

Interactive plots

These examples show how to use Shiny's interactive plotting features

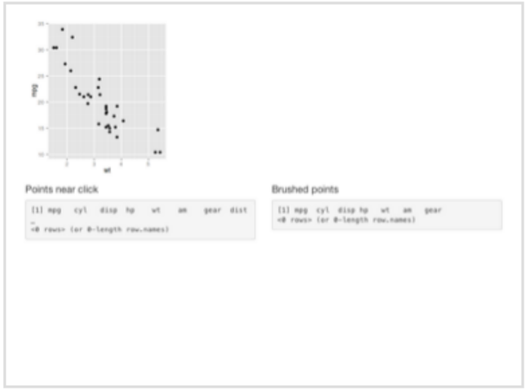


Plot interaction - basic

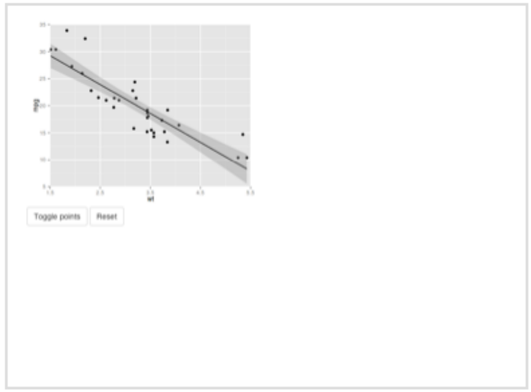


Plot interaction - advanced

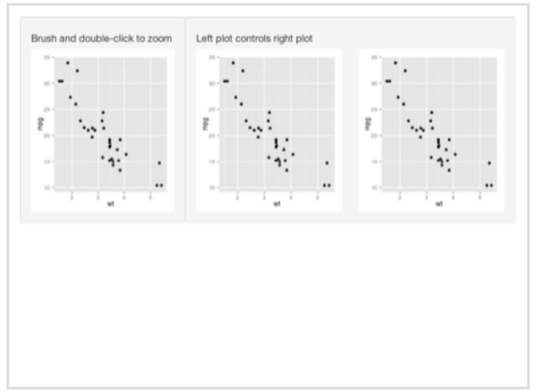
Image interaction - basic



Plot interaction - selecting points



Plot interaction - exclude



Plot interaction - zoom

plotOutput()

To collect input values, add **click**, **dblclick**, **hover**, or **brush** arguments.

```
plotOutput(…, click = "myclick")
```

plotOutput()

To collect input values, add **click**, **dblclick**, **hover**, or **brush** arguments.

```
plotOutput(…, click = "myclick")
```

stores value as

```
input$myclick
```

Your Turn

Run the app on the following slide or this fancier version:

- shiny.calvin.edu/rpruim/ShinyDemos/InteractivePlots/

Then explore how the values of

- `clicked`,
- `dblclicked`,
- `hovered`, and
- `brushed`

change as you manipulate the plot with you mouse.

```
ui <- fluidPage(
  plotOutput("plot", click = "click", dblclick = "dblclick",
    hover = "hover", brush = "brush"),
  fluidRow(
    column(3, "Clicked",        verbatimTextOutput("clicked")),
    column(3, "Double Clicked", verbatimTextOutput("dblclicked")),
    column(3, "Hovered",        verbatimTextOutput("hovered")),
    column(3, "Brushed",        verbatimTextOutput("brushed"))
  ))
server <- function(input, output) {
  output$plot       <- renderPlot(qplot(wt, mpg, data = mtcars))
  output$clicked    <- renderPrint(input$click)
  output$dblclicked <- renderPrint(input$dblclick)
  output$hovered    <- renderPrint(input$hover)
  output$brushed    <- renderPrint(input$brush)
}
shinyApp(ui, server)
```

fancy version @ shiny.calvin.edu/rpruim/ShinyDemos/InteractivePlots/

plotOutput()

Location of mouse click (in x and y coordinates)

Location of double click (in x and y coordinates)

Location of stationary mouse (in x and y)

Bounding coordinates of brush box (in x and y)

```
plotOutput(…,
  click = "click",
  dblclick = "dblclick",
  hover = "hover",
  brush = "brushed"
)
```

nearPoints()

Returns a data frame of points near a click

```
nearPoints(mtcars, input$click, xvar = "wt",
           yvar = "mpg", threshold = 5)
```

nearPoints()

Returns a data frame of points near a click

data frame to return subset of (should match plot)

```
nearPoints(mtcars, input$click, xvar = "wt",
           yvar = "mpg", threshold = 5)
```

nearPoints()

Returns a data frame of points near a click

data frame to return subset of (should match plot)

click input object

```
nearPoints(mtcars, input$click, xvar = "wt",
           yvar = "mpg", threshold = 5)
```

nearPoints()

Returns a data frame of points near a click

data frame to return subset of (should match plot)

click input object

x variable in plot (not needed with ggplot2)

```
nearPoints(mtcars, input$click, xvar = "wt",
           yvar = "mpg", threshold = 5)
```

nearPoints()

Returns a data frame of points near a click

data frame to return subset of (should match plot)

click input object

x variable in plot (not needed with ggplot2)

```
nearPoints(mtcars, input$click, xvar = "wt",
           yvar = "mpg", threshold = 5)
```

y variable in plot (not needed with ggplot2)

nearPoints()

Returns a data frame of points near a click

data frame to return subset of (should match plot)

click input object

x variable in plot (not needed with ggplot2)

```
nearPoints(mtcars, input$click, xvar = "wt",
           yvar = "mpg", threshold = 5)
```

y variable in plot (not needed with ggplot2)

include points that fall within this many pixels of click

brushedPoints()

Returns a data frame of points within a brushed area

```
brushedPoints(mtcars, input$brush,
              xvar = "wt", yvar = "mpg")
```

brushedPoints()

Returns a data frame of points within a brushed area

data frame to return subset of (should match plot)

```
brushedPoints(mtcars, input$brush,
              xvar = "wt", yvar = "mpg")
```

brushedPoints()

Returns a data frame of points within a brushed area

data frame to return subset of (should match plot)

brush input object

```
brushedPoints(mtcars, input$brush,
              xvar = "wt", yvar = "mpg")
```

brushedPoints()

Returns a data frame of points within a brushed area

data frame to return subset of (should match plot)

brush input object

```
brushedPoints(mtcars, input$brush,
              xvar = "wt", yvar = "mpg")
```

x variable in plot
(not needed with ggplot2)

brushedPoints()

Returns a data frame of points within a brushed area

data frame to return subset of (should match plot)

brush input object

```
brushedPoints(mtcars, input$brush,
              xvar = "wt", yvar = "mpg")
```

x variable in plot
(not needed with ggplot2)

y variable in plot
(not needed with ggplot2)

Learn more

shiny.rstudio.com/articles

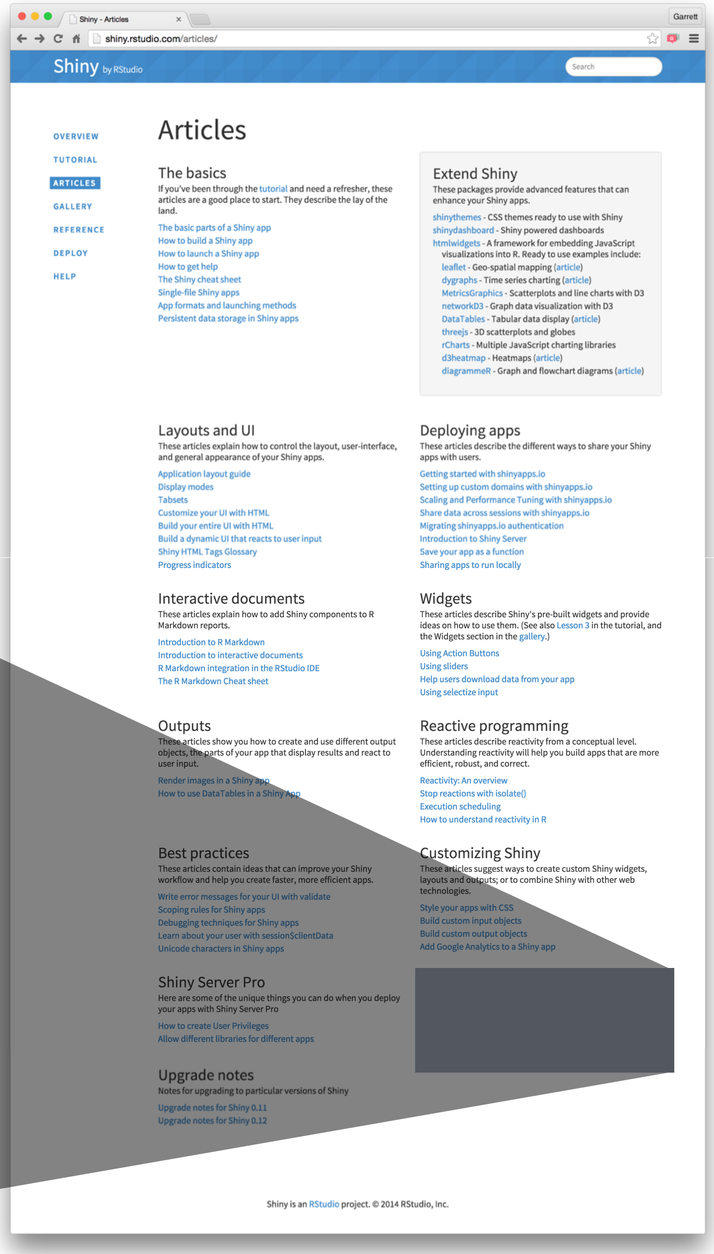
Interactive plots

Create interactive plots with base and ggplot2 graphics

Interactive plots

Selecting rows of data

Interactive plots - advanced



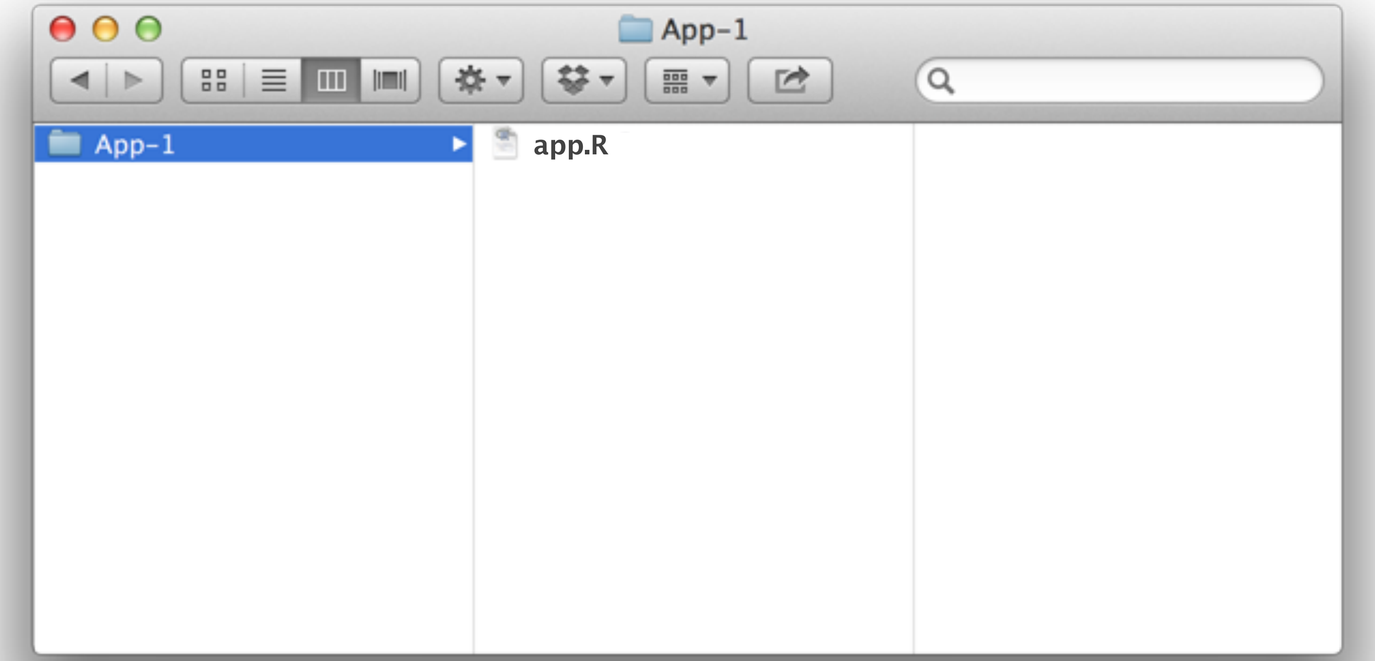
Share

your app

How to save your app

One directory with every file the app needs:

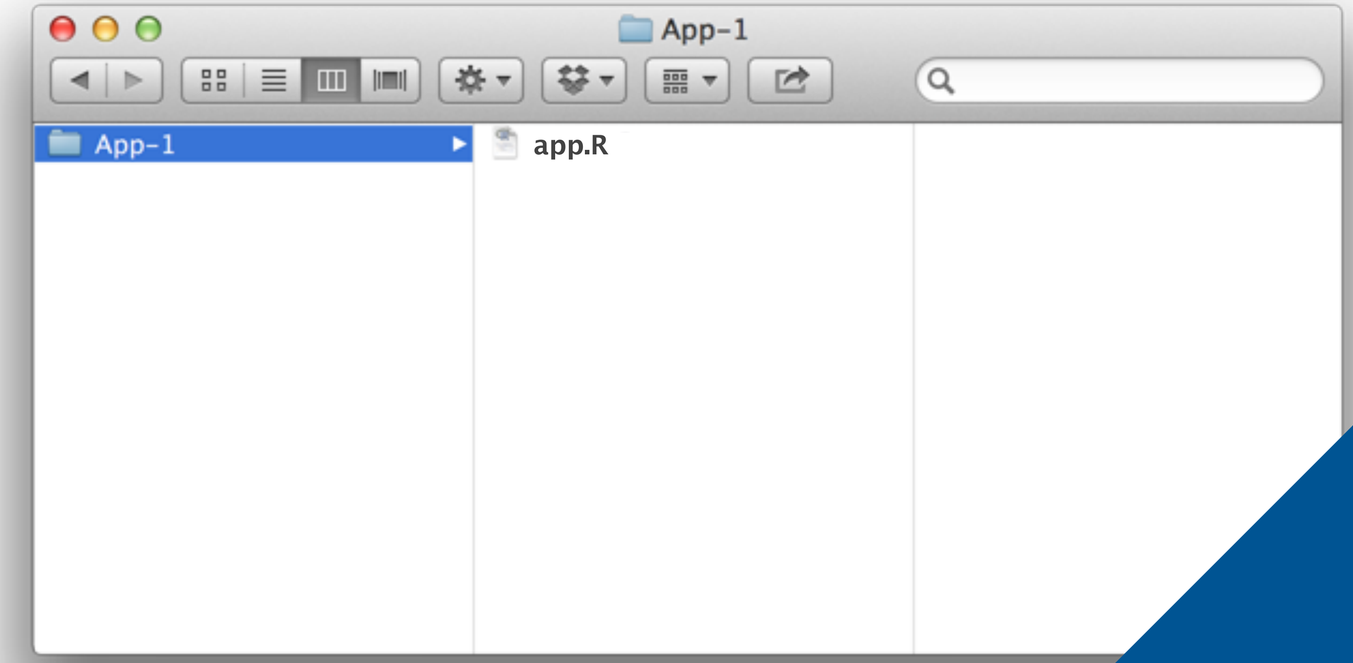
- app.R *(your script which ends with a call to shinyApp())*
- datasets, images, css, helper scripts, etc.



How to save your app

One directory with every file the app needs:

- app.R *(your script which ends with a call to shinyApp())*
- datasets, images, css, helper scripts, etc.



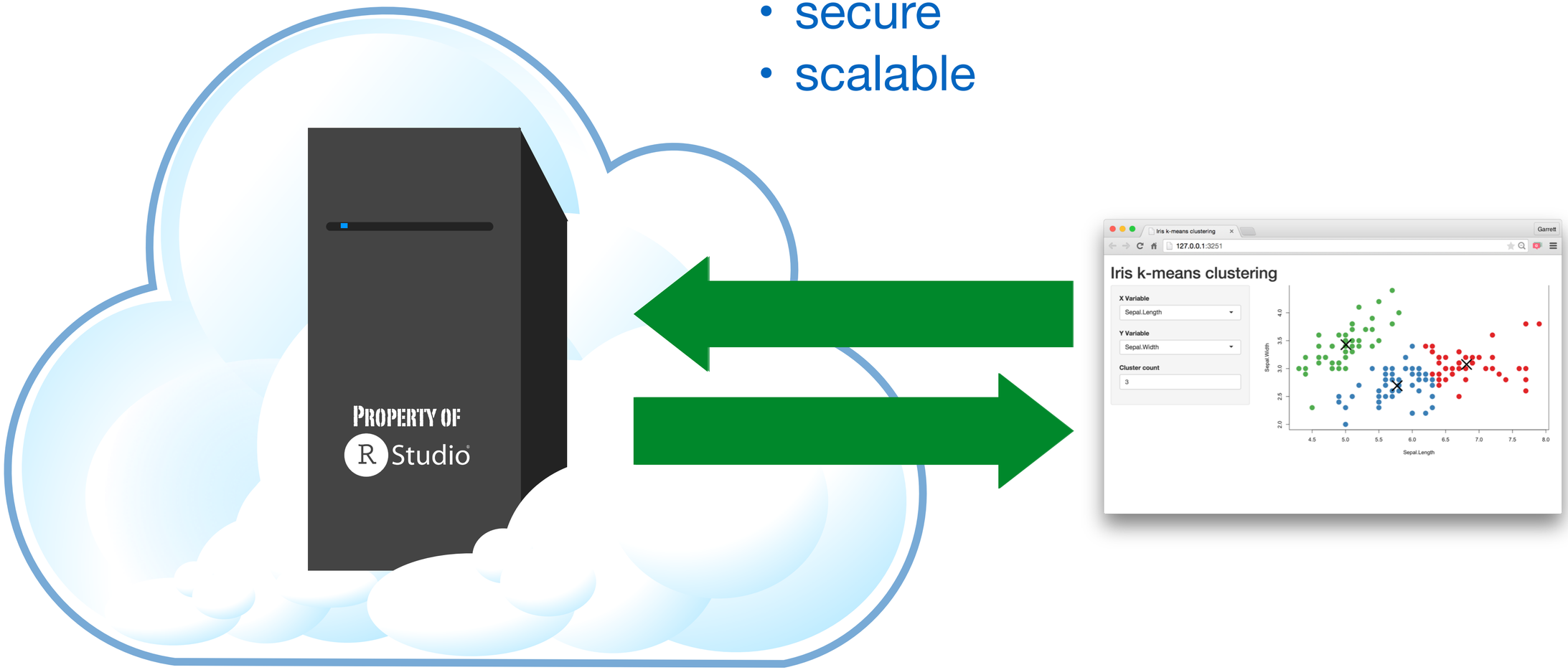
You must use this exact name (**app.R**)

shinyapps.io

Shinyapps.io

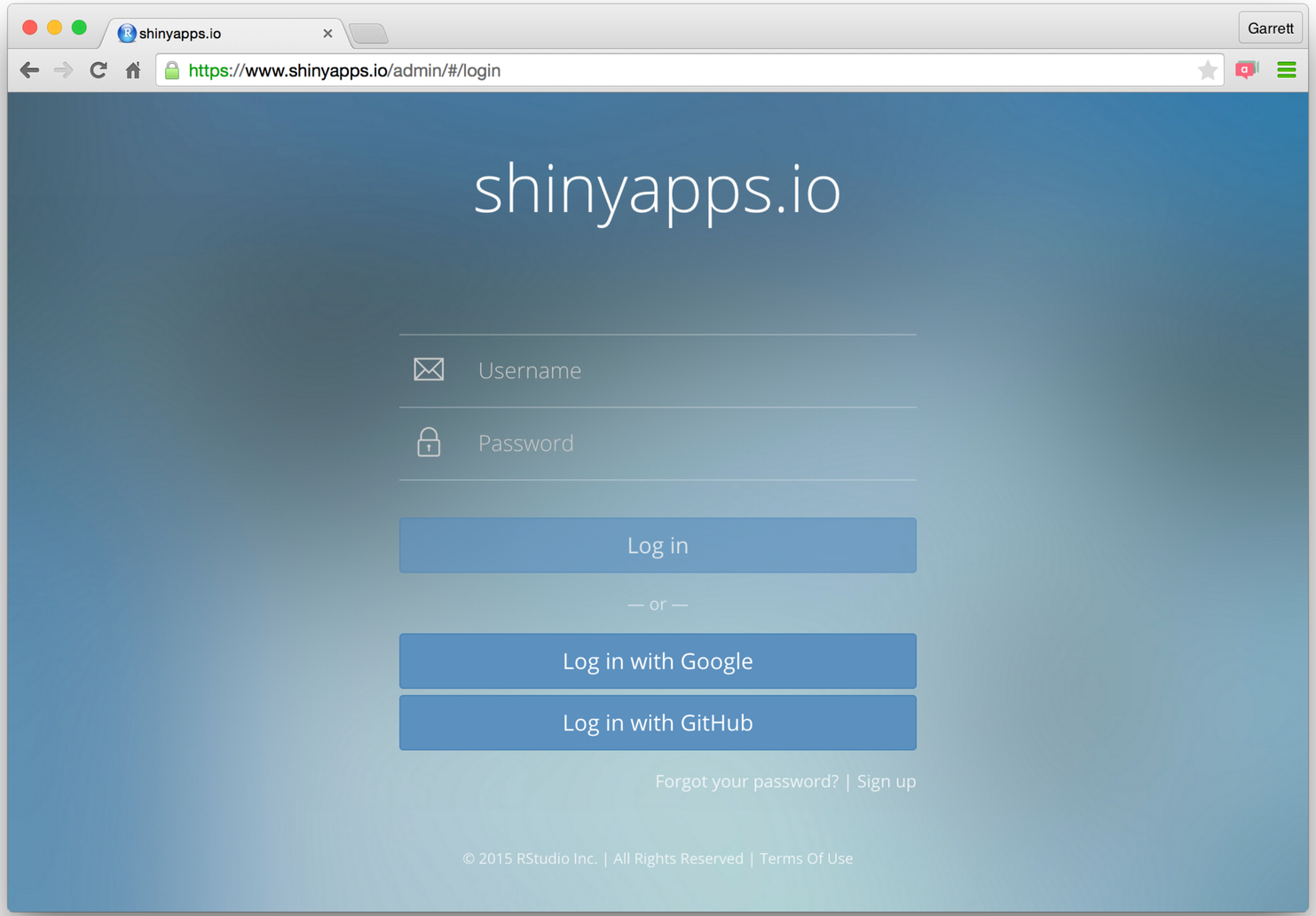
A server maintained by RStudio

- easy to use
- secure
- scalable



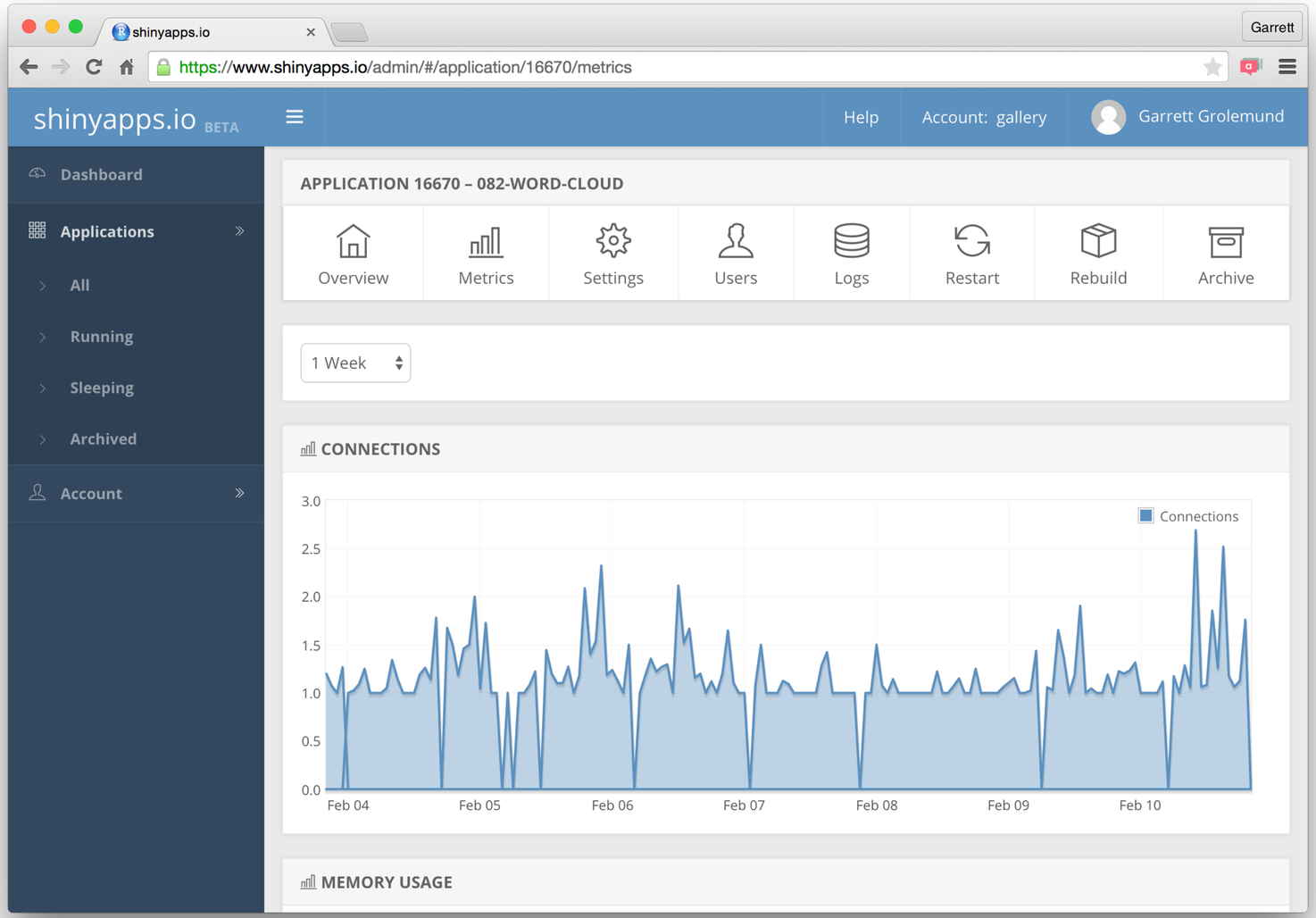
Hassle-free cloud hosting for Shiny

www.shinyapps.io



Hassle-free cloud hosting for Shiny

shinyapps.io



FREE

$0 /month

New to Shiny? Deploy your applications to the cloud for FREE. Perfect for teachers and students or those who want a place to learn and play. No credit card required.

- **5** Applications
- **25** Active Hours
- Community Support
- RStudio Branding

BASIC

$39 /month

(or $440/year)

Take your users' experience to the next level. shinyapps.io Basic lets you scale your application performance by adding R processes dynamically as usage increases.

- **Unlimited** Applications
- **250** Active Hours
- Multiple Instances
- Email Support

STANDARD

$99 /month

(or $1,100/year)

Need password protection? shinyapps.io Standard lets you authenticate your application users.

- **Unlimited** Applications
- **1000** Active Hours
- Authentication
- Multiple Instances
- Email Support

PROFESSIONAL

$299 /month

(or $3,300/year)

shinyapps.io Professional has it all. Share an account with others in your business or change your shinyapps.io domain into a URL of your own.

- **Unlimited** Applications
- **5000** Active Hours
- Authentication
- Multiple Users
- Multiple Instances
- Custom Domains*
- Email Support

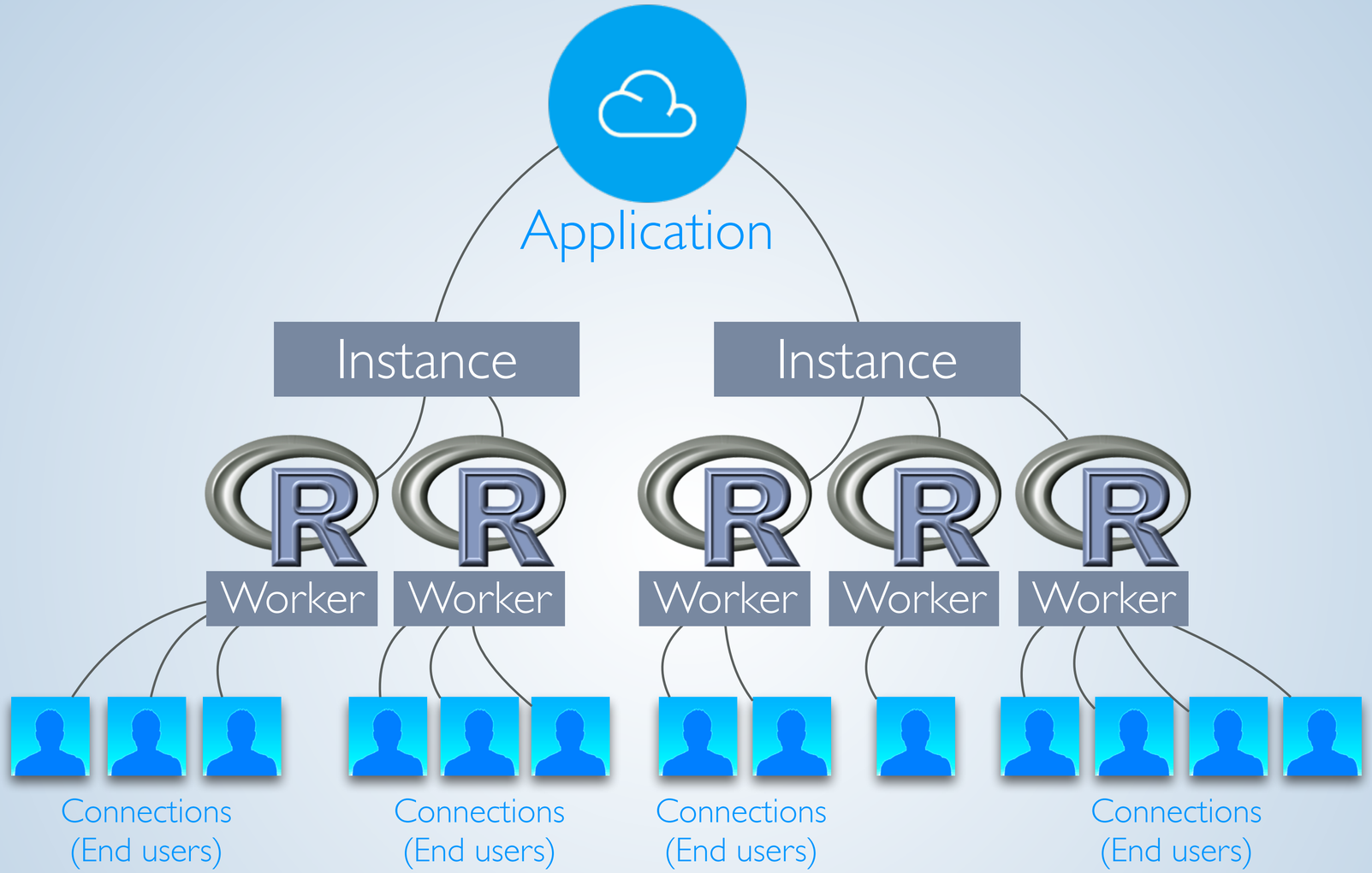
Build your own server

Shiny Server Pro

www.rstudio.com/products/shiny/shiny-server/

- **Secure access**
  LDAP, GoogleAuth, SSL, and more
- **Performance**
  fine tune at app and server level
- **Management**
  monitor and control resource use
- **Support**
  direct priority support

45 day evaluation free trial


Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

Application

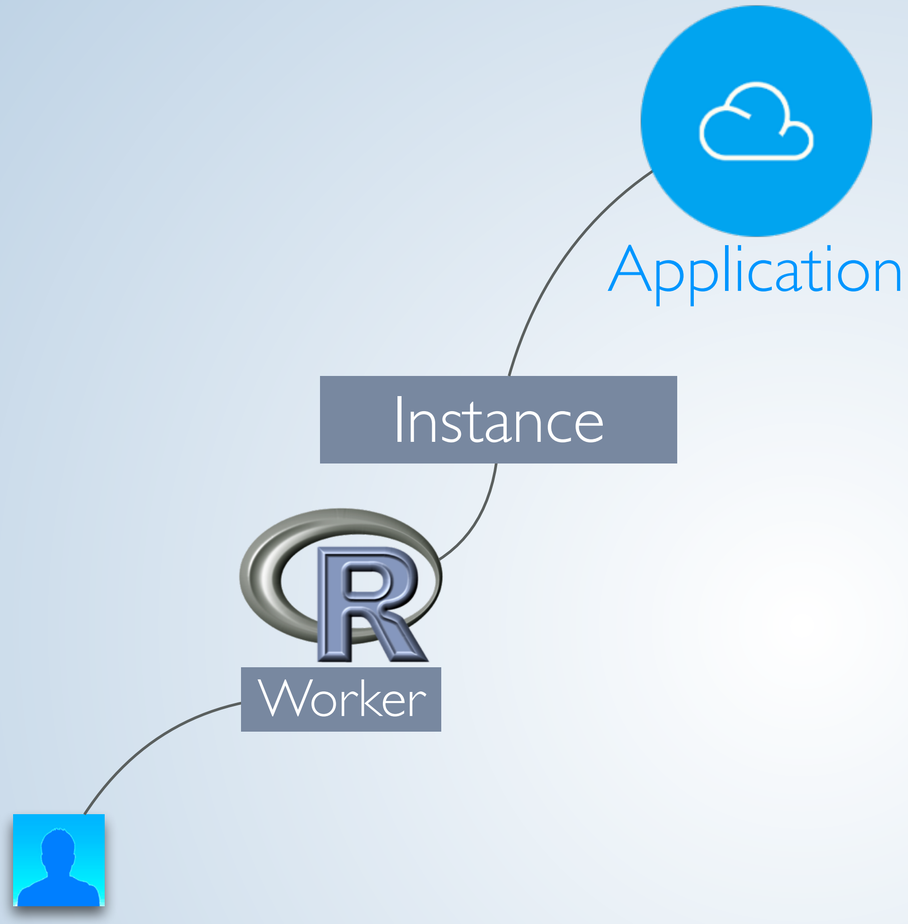
Connections
(End users)

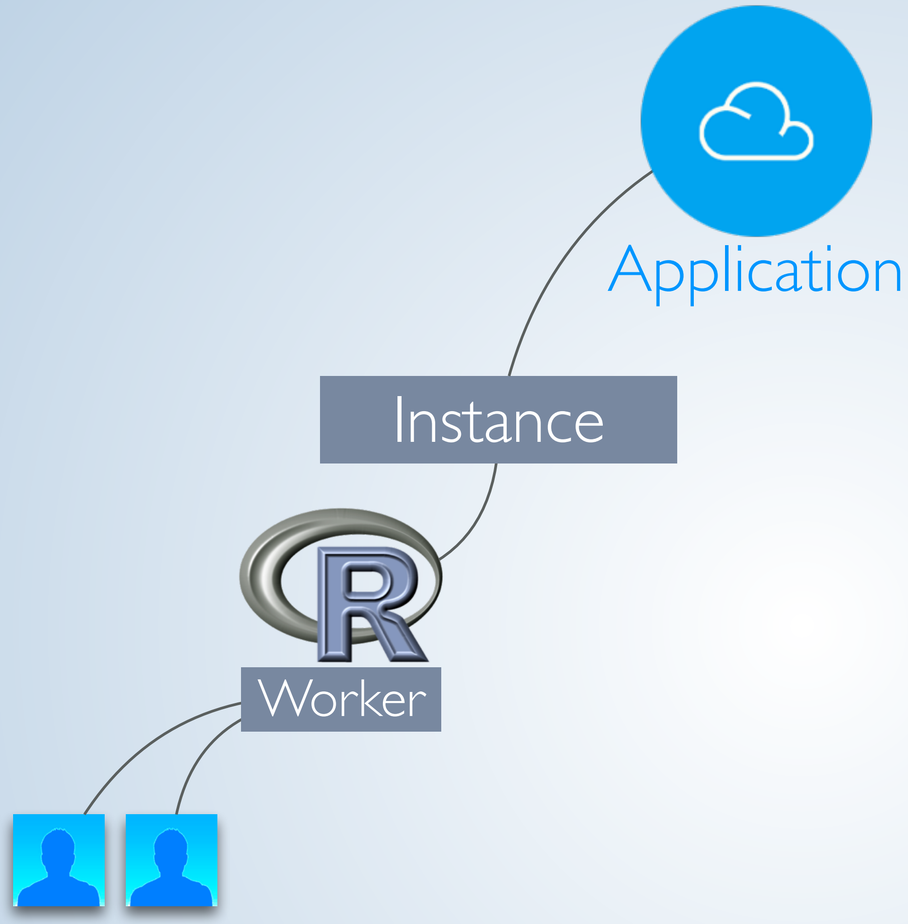
Connections
(End users)

Connections
(End users)

Connections
(End users)

RStudio®


Application
Instance
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)



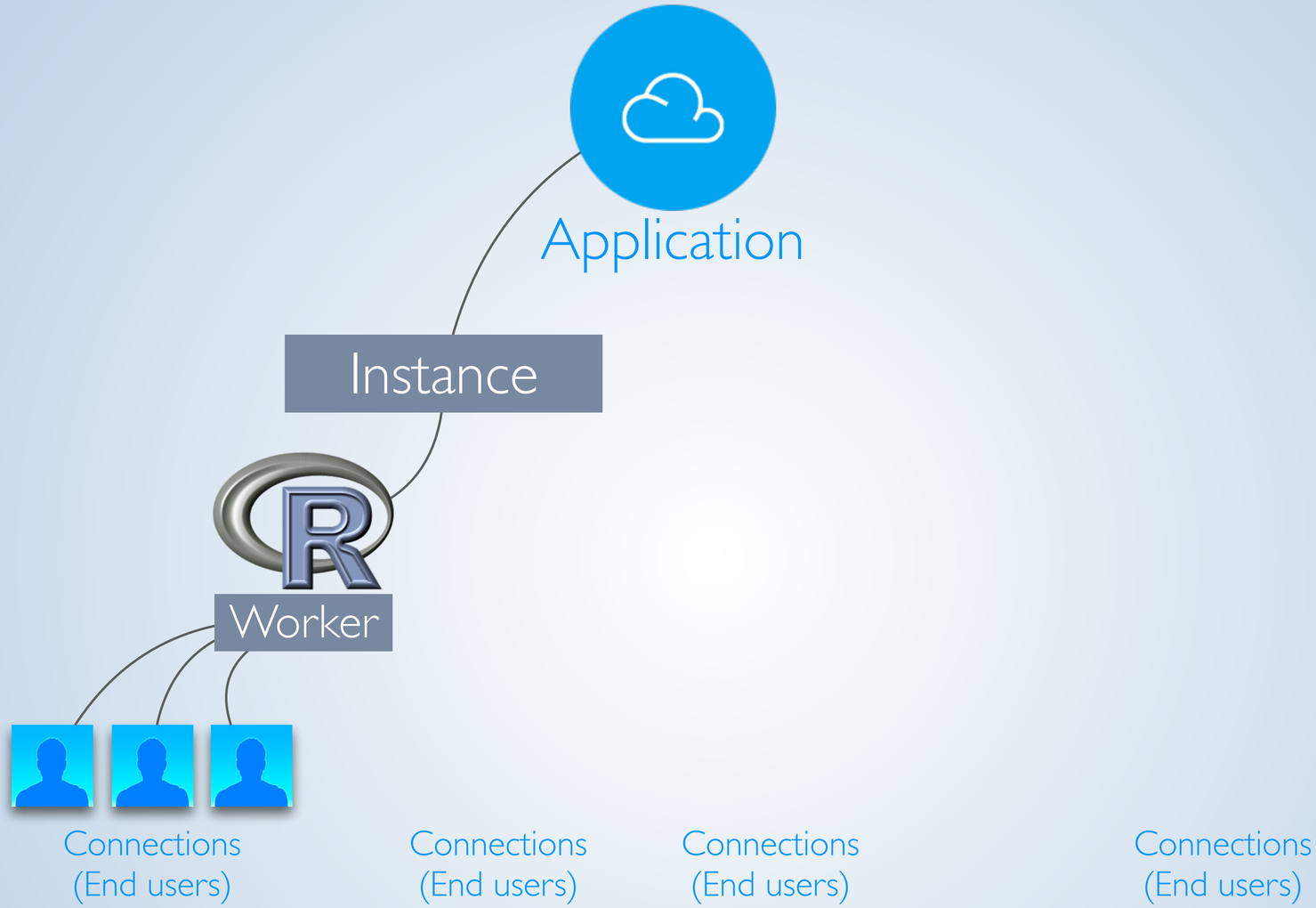
Connections
(End users)

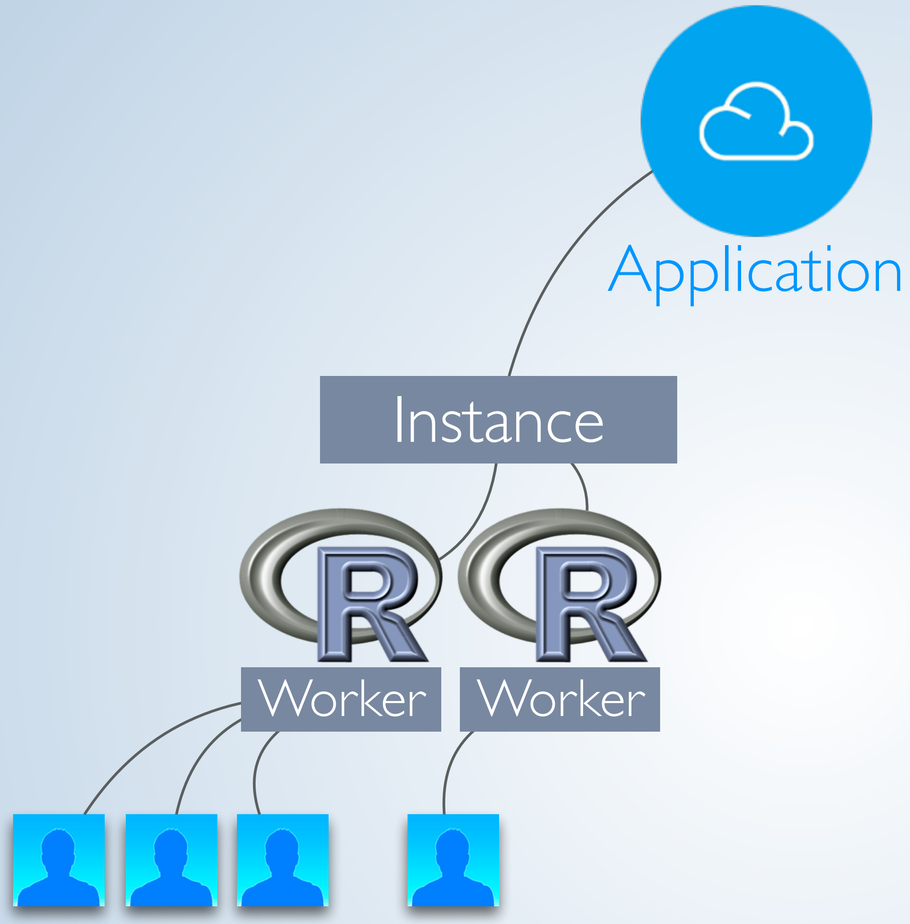
Connections
(End users)

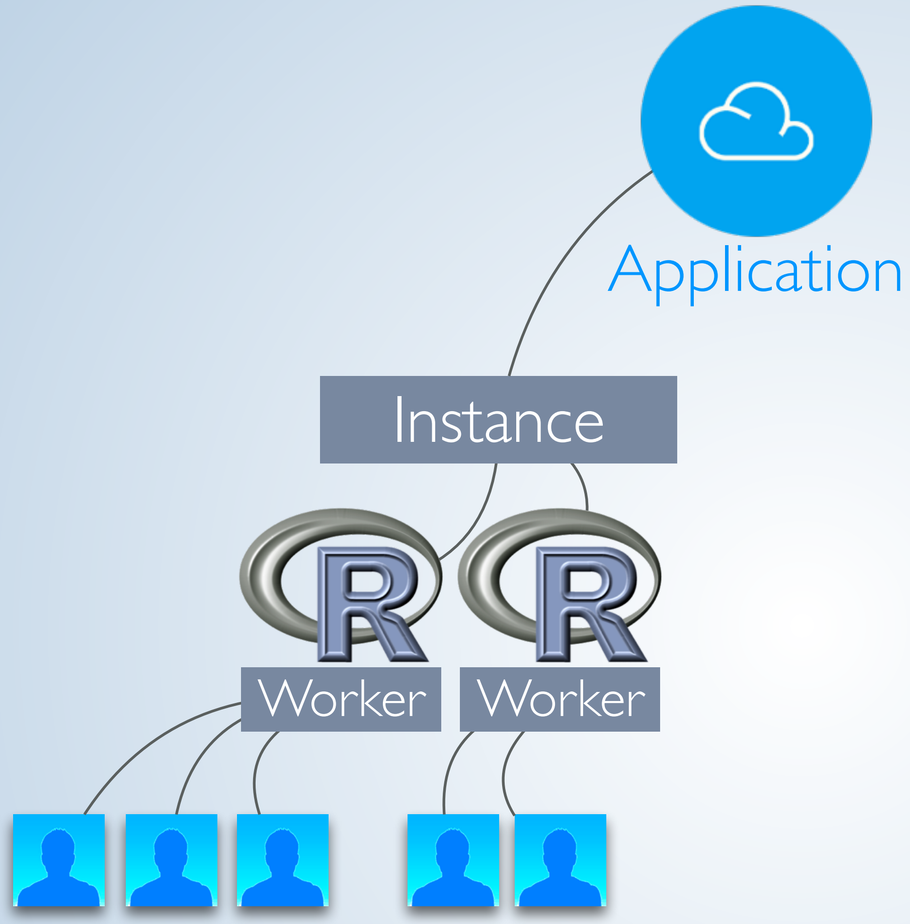
Connections
(End users)

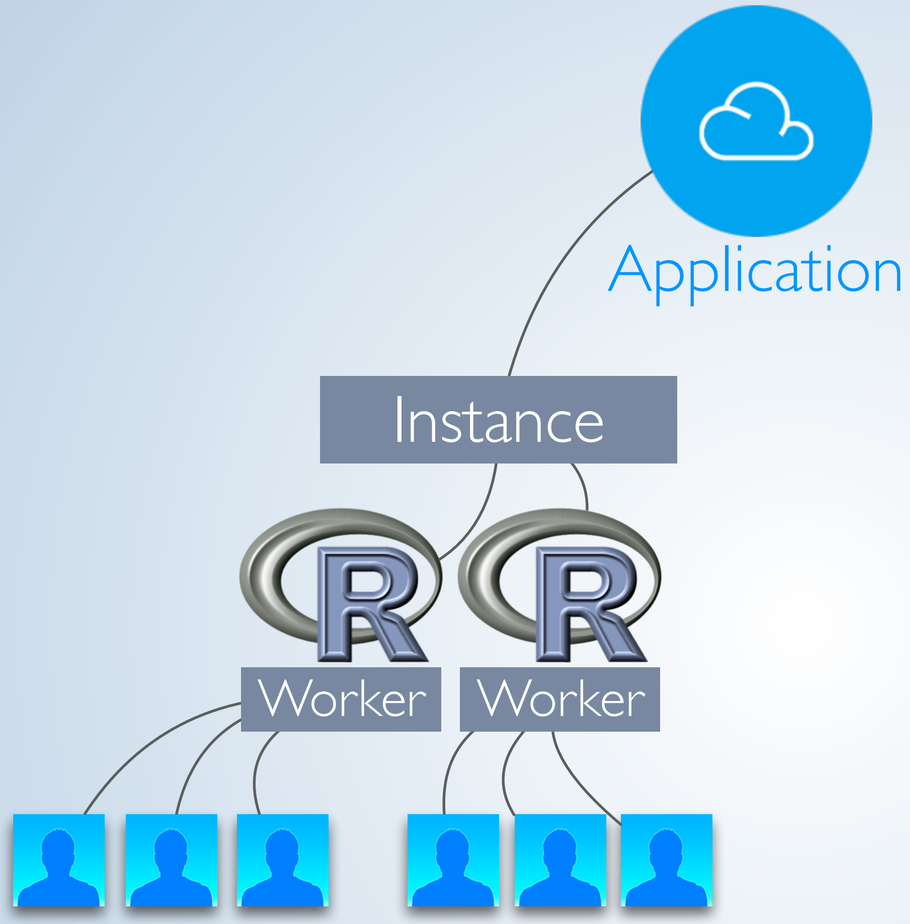
Connections
(End users)

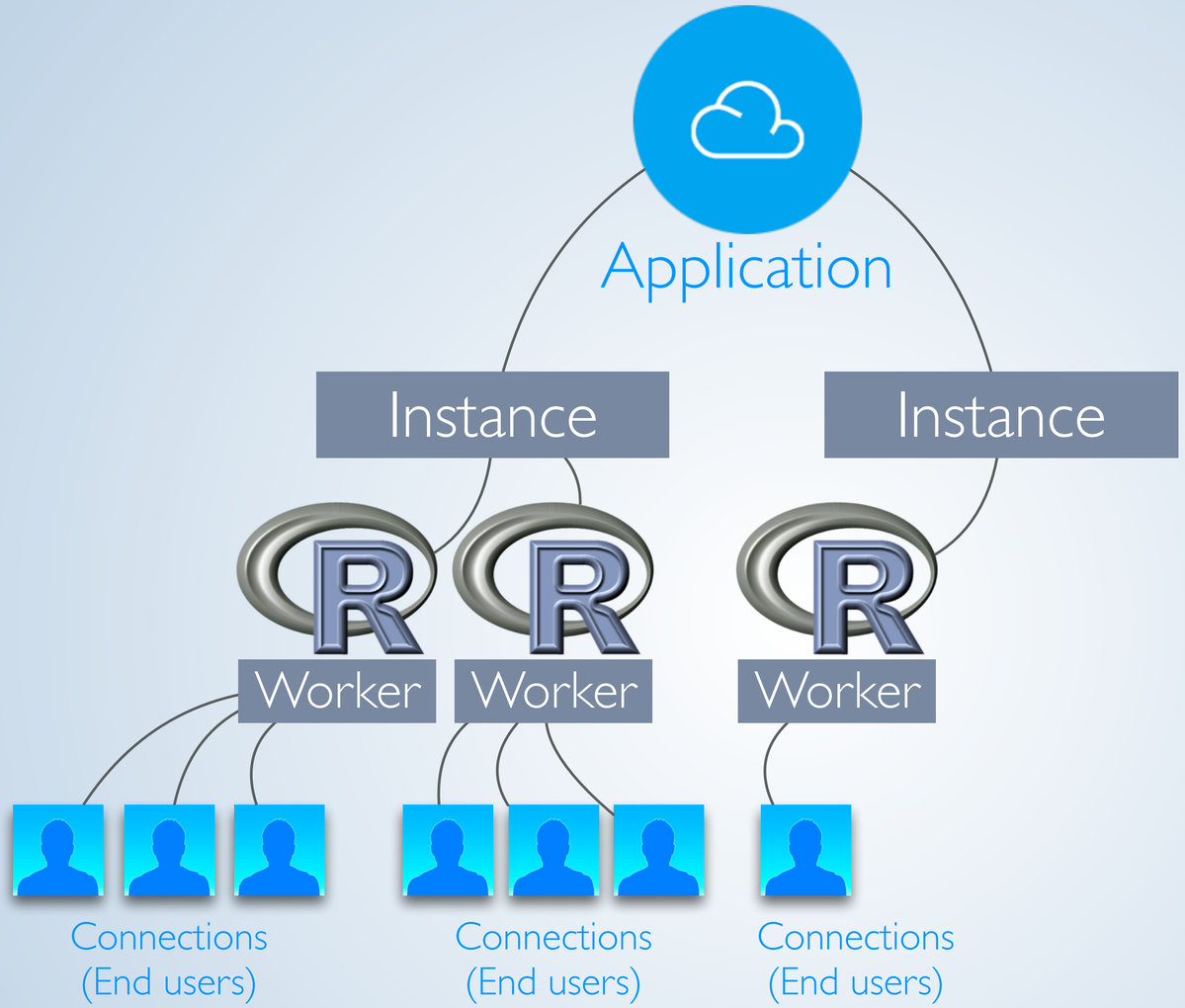
RStudio®

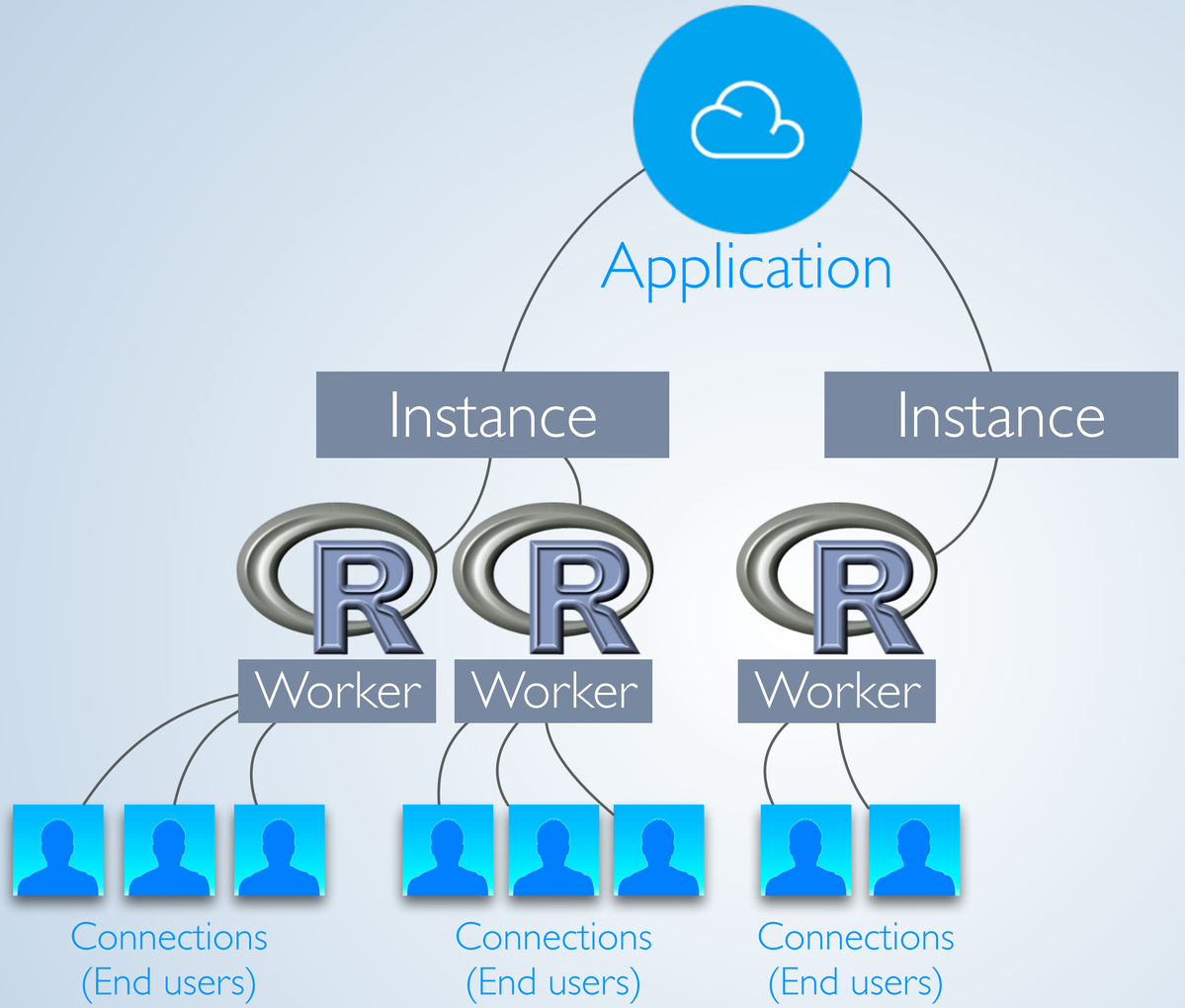

Application
Instance
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

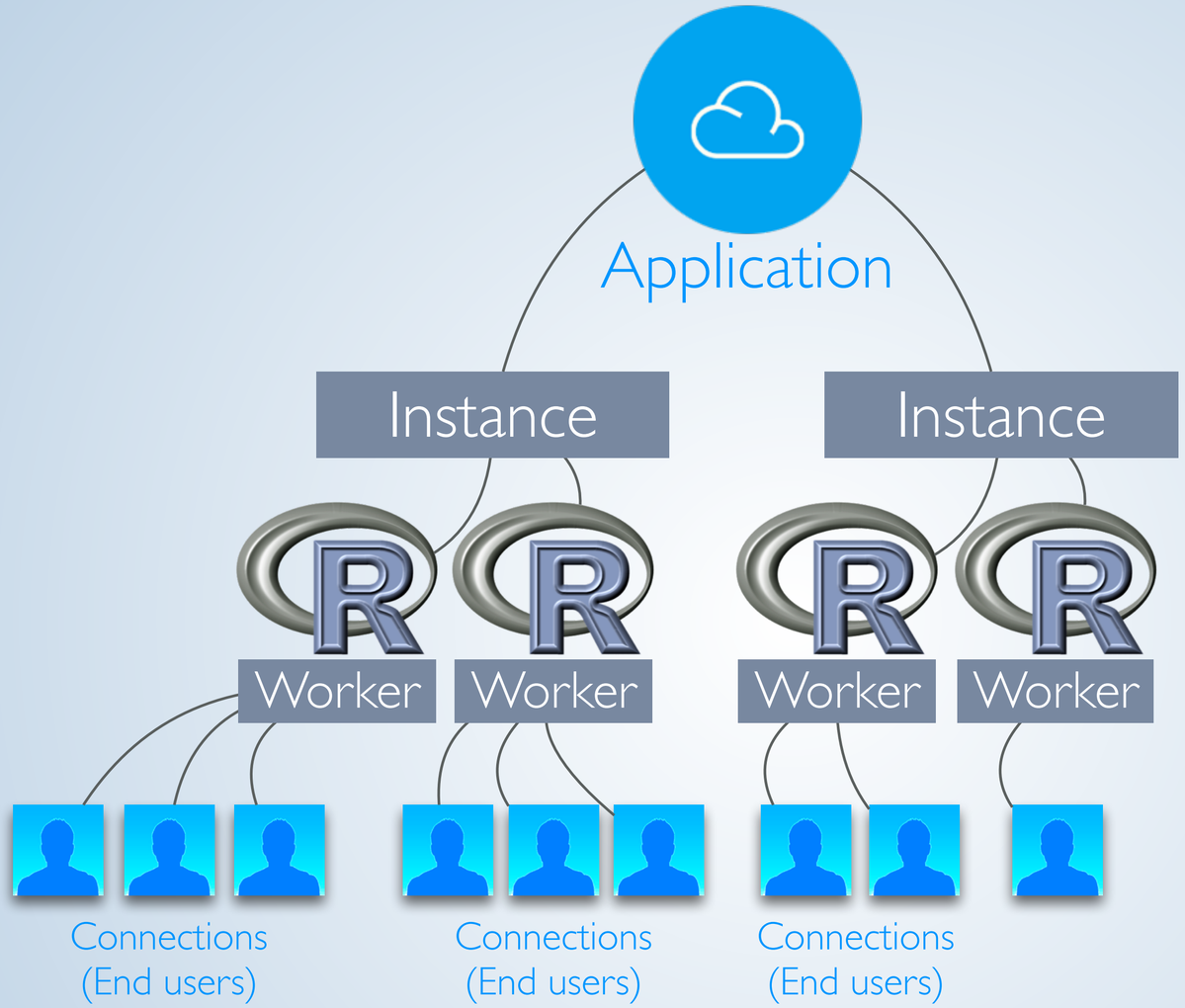

Application
Instance
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

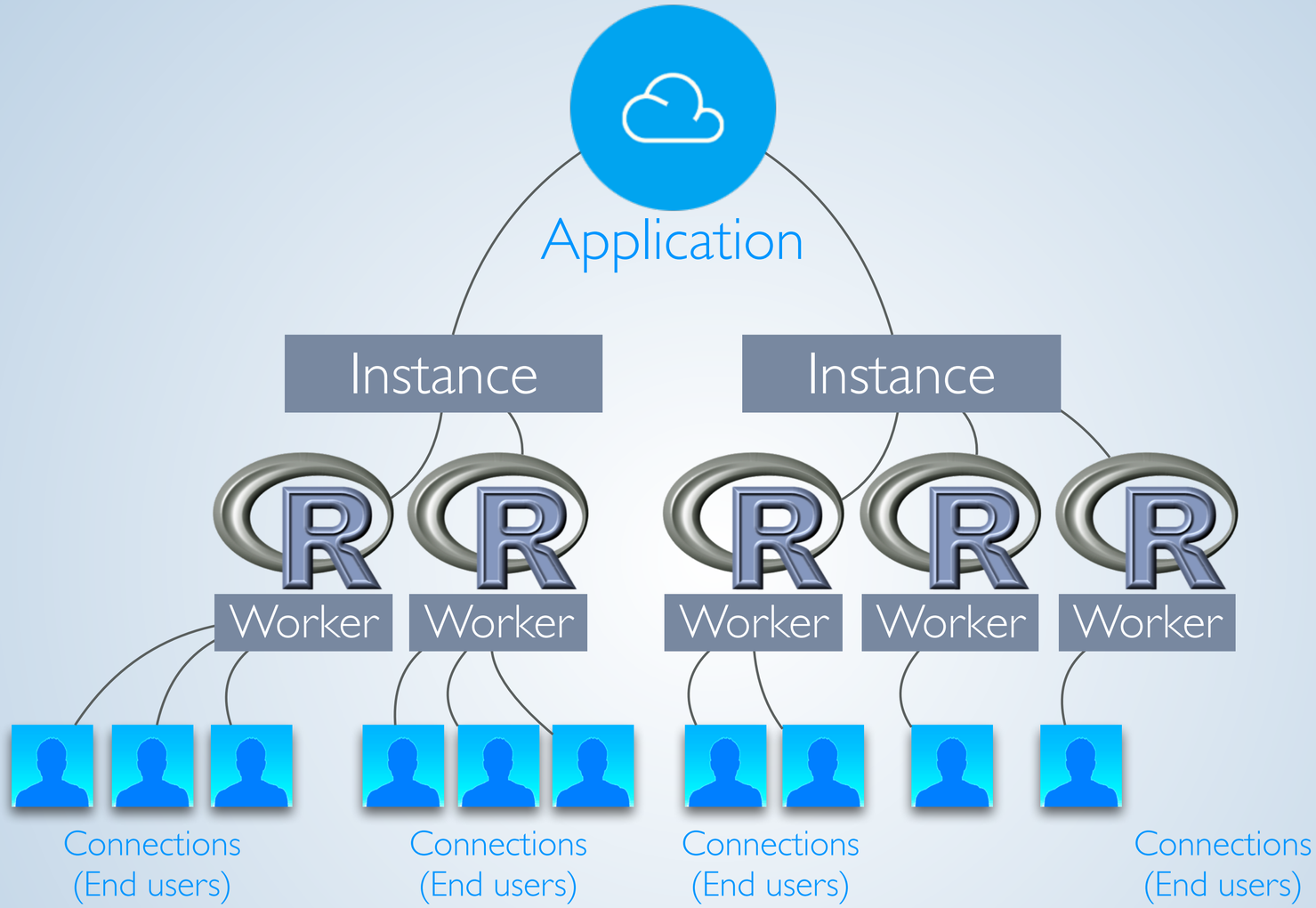

Application
Instance
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

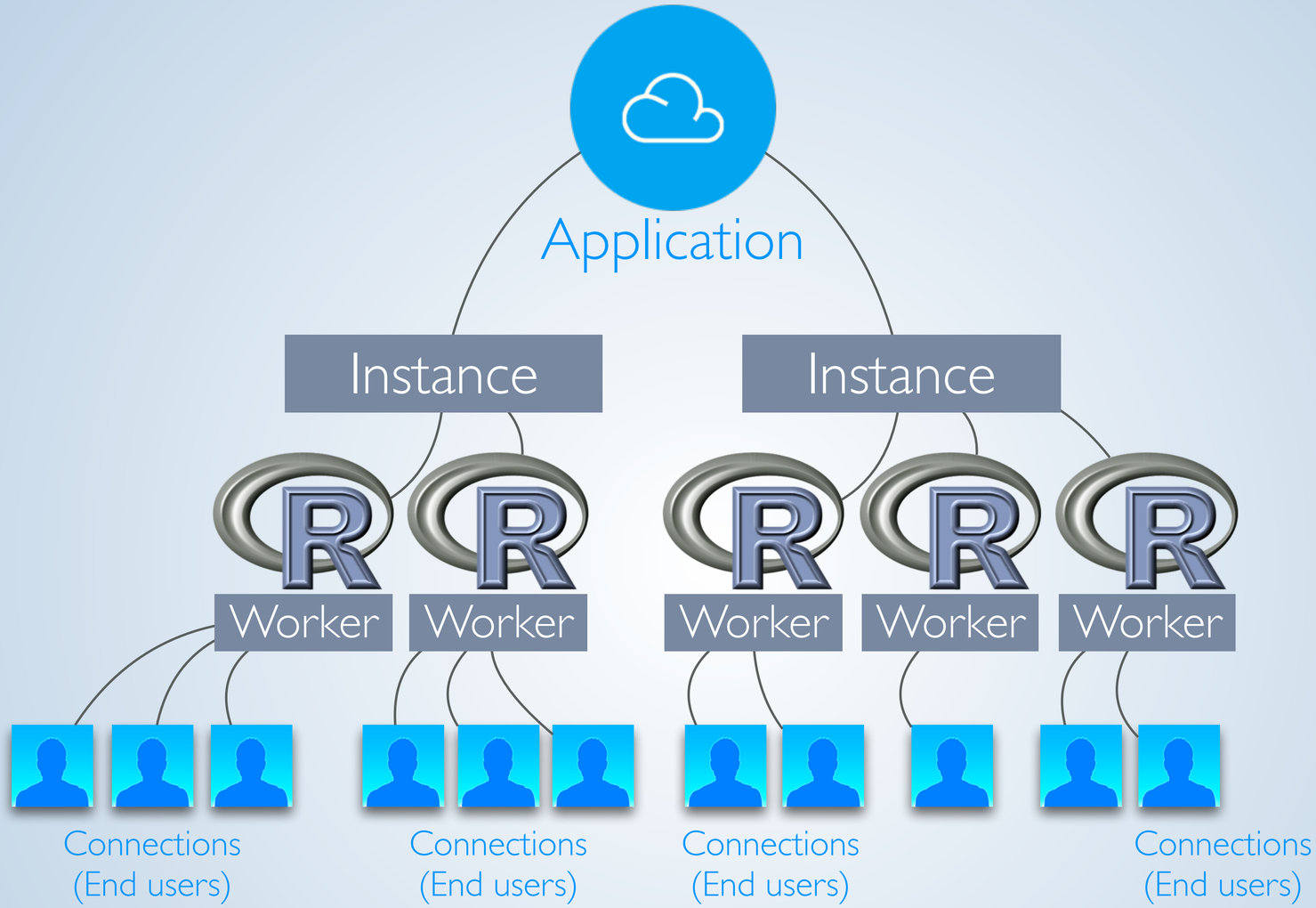

Application
Instance
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)


Application
Instance
Instance
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

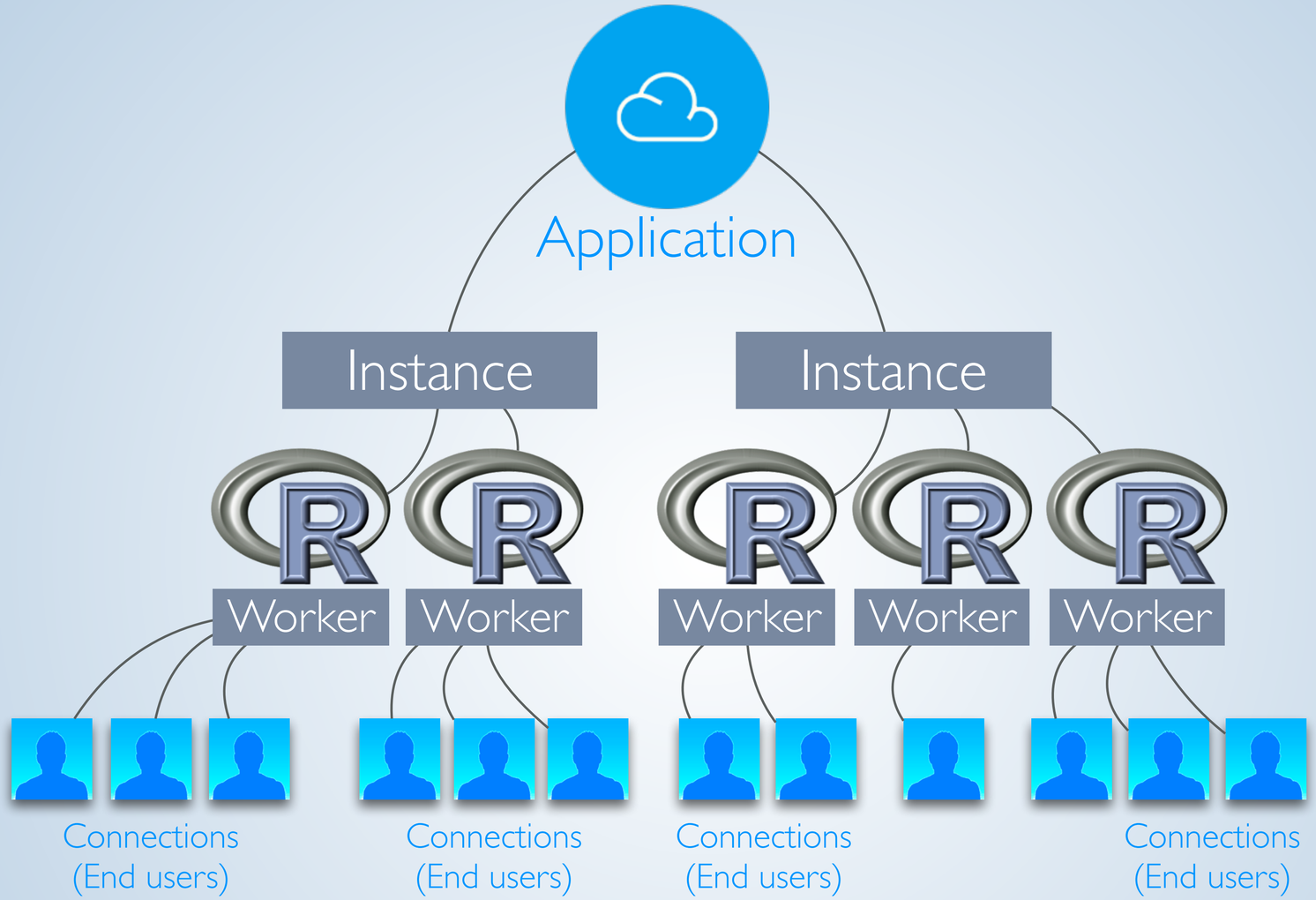

Application
Instance
Instance
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

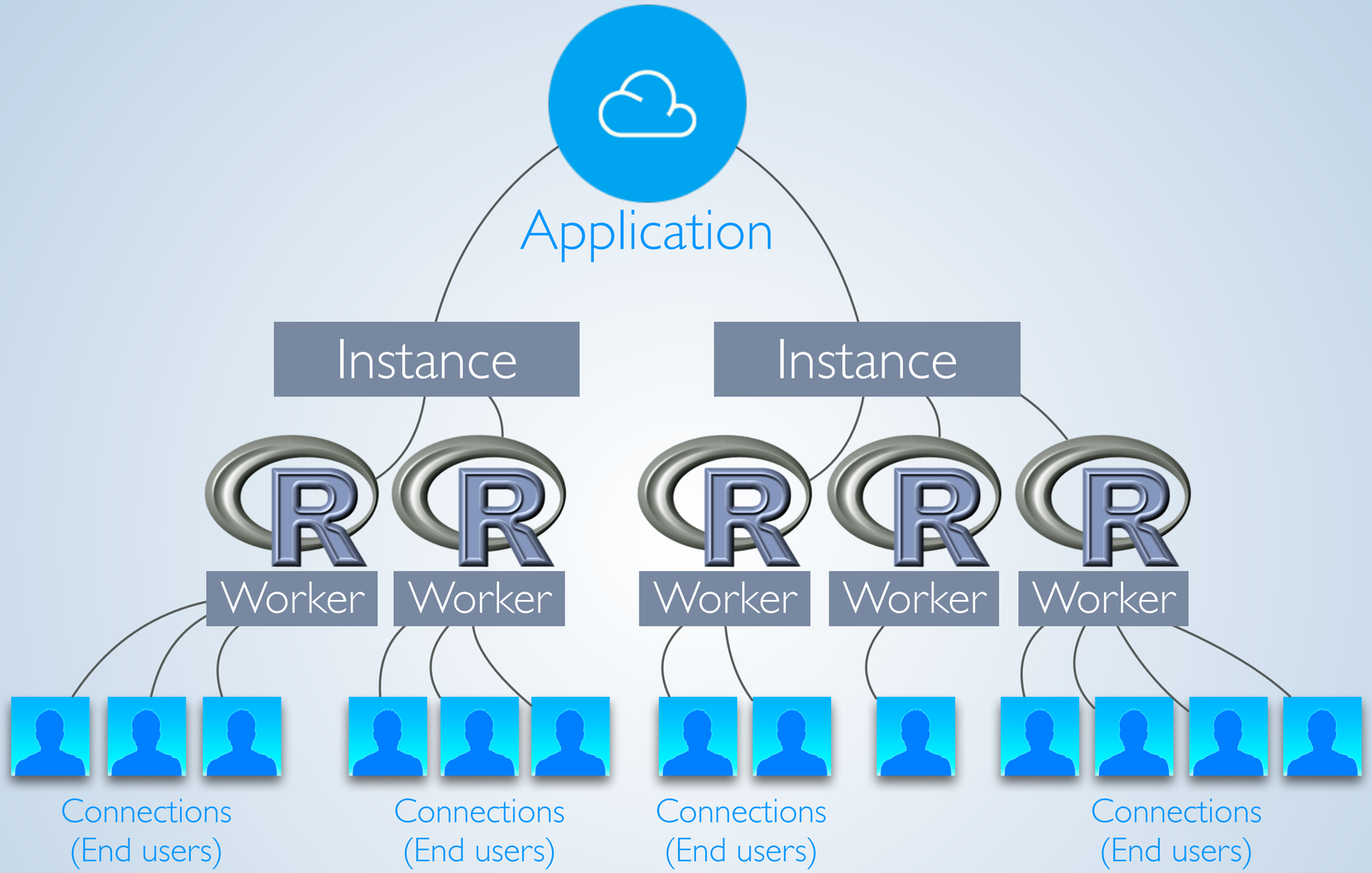

Application
Instance
Instance
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

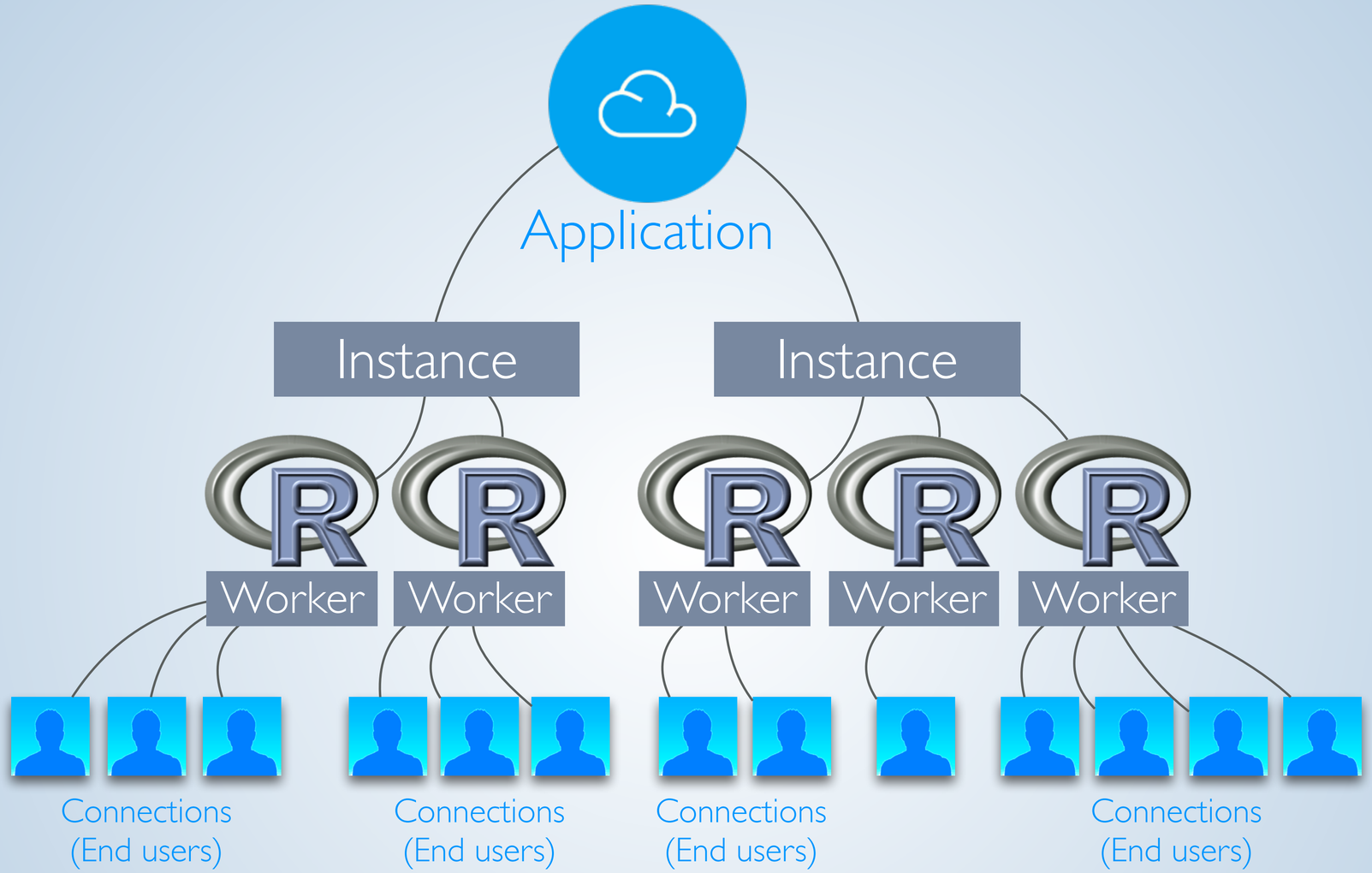

Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)


Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

RStudio®


Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)


Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)


Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

Application

Connections
(End users)

Connections
(End users)

Connections
(End users)

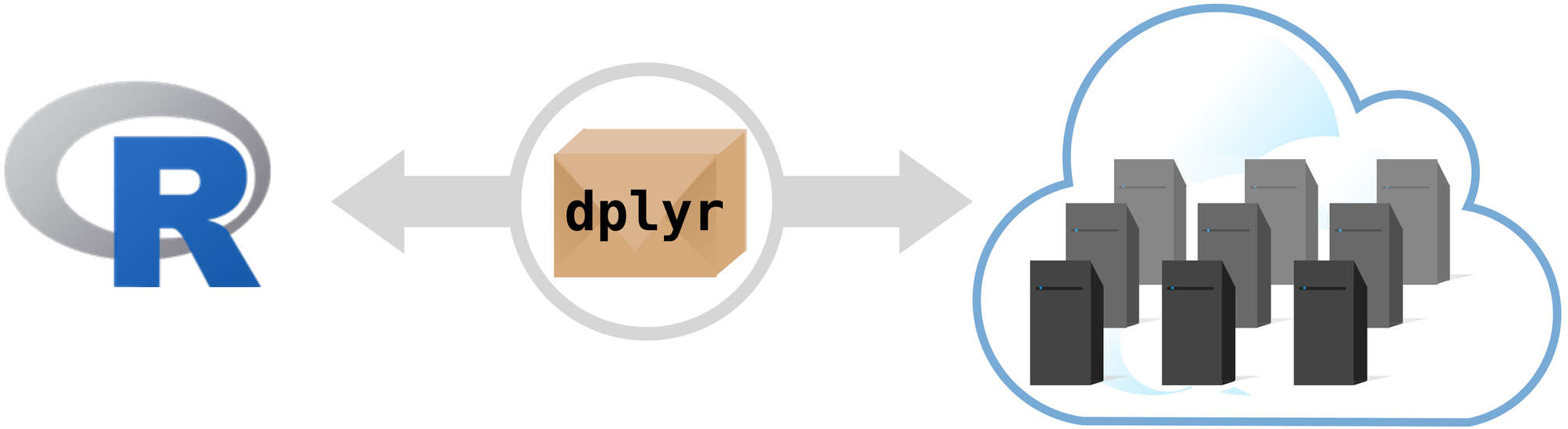
Connections
(End users)

RStudio®

Advice for Big Data

General Strategy

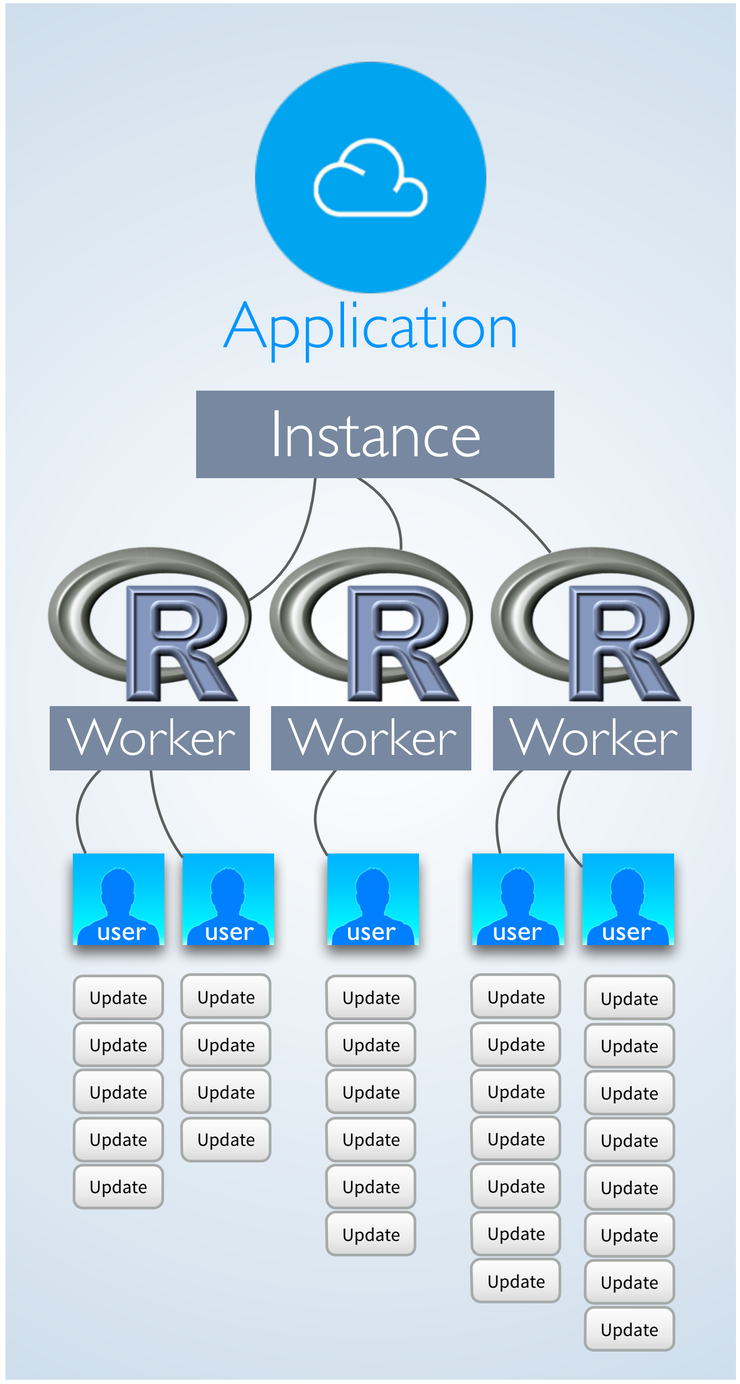
1. Store data in out of memory warehouse
2. Use an R Package to interact with warehouse



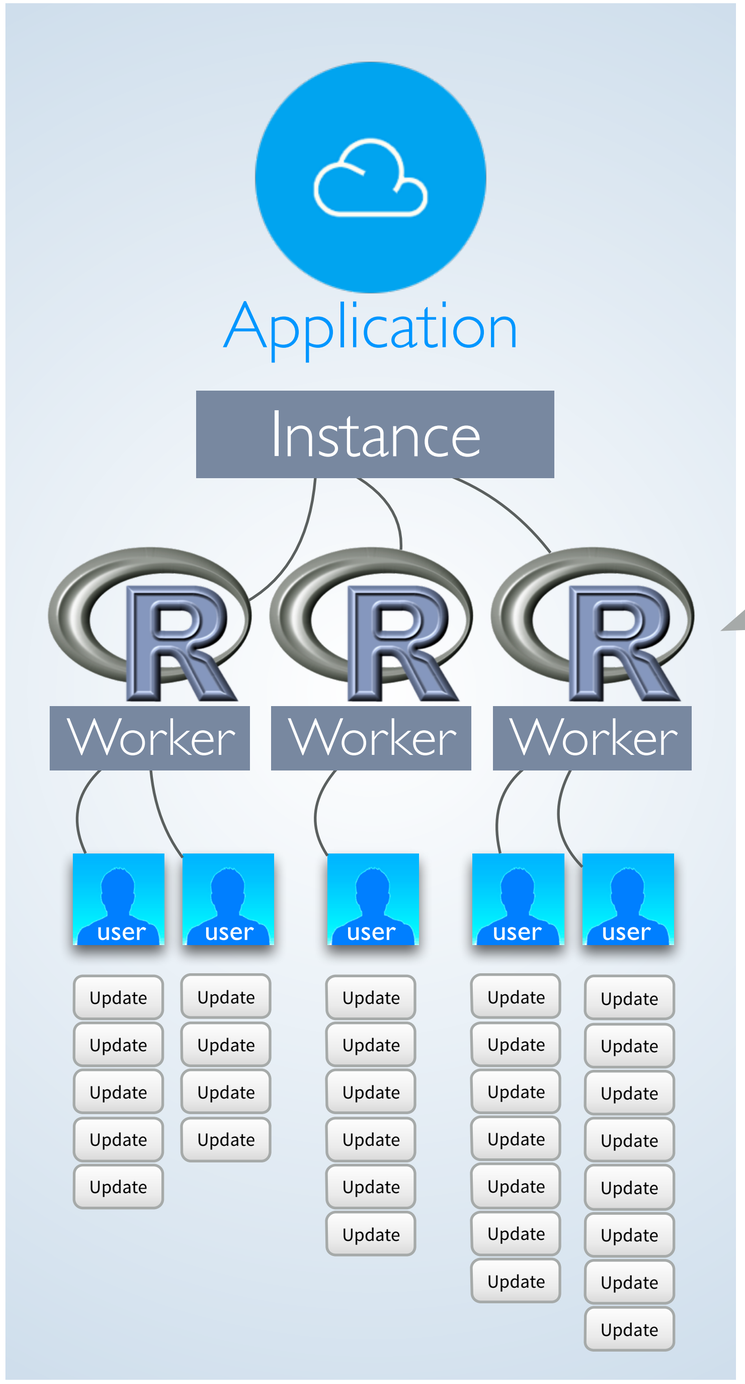
Big Data and Shiny

1. **Avoid** unnecessary repetitions

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```

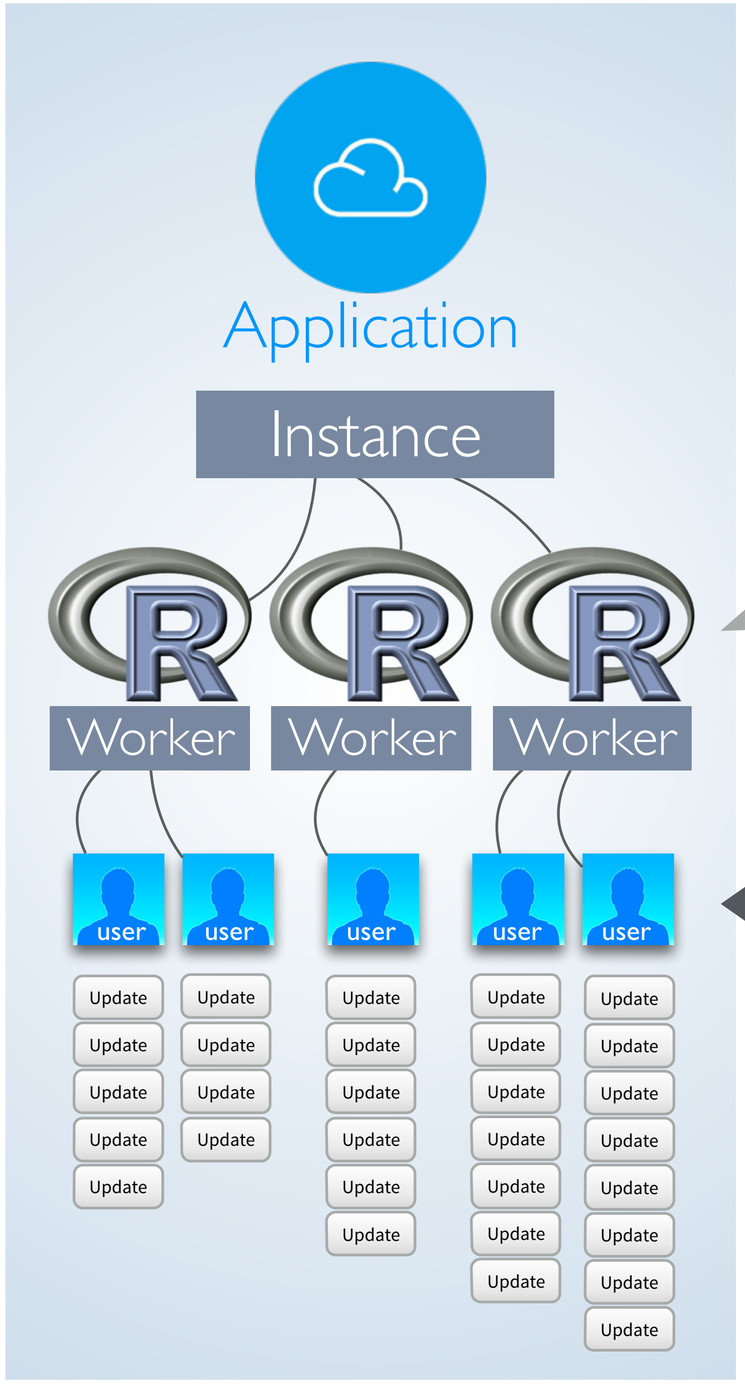



```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



Code outside the server function will be run once per R worker

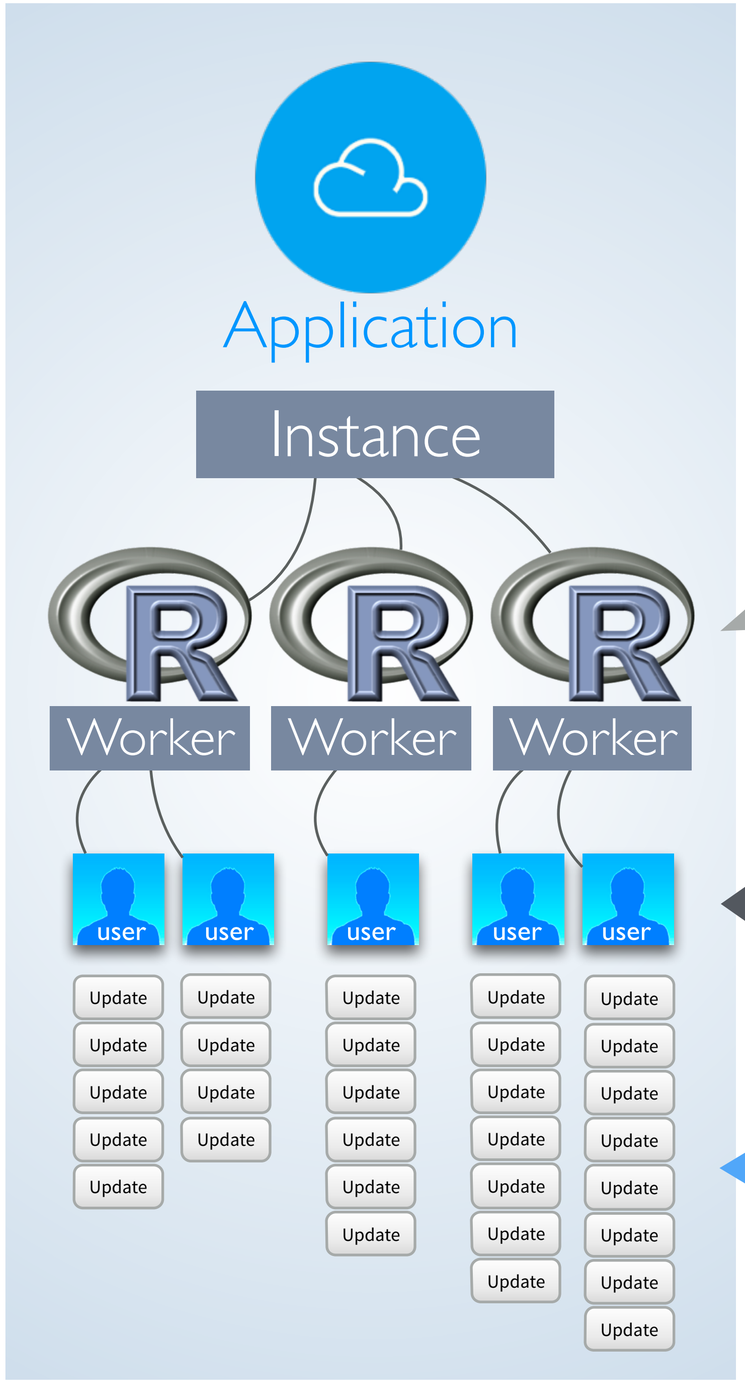
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



Code outside the server function will be run once per R worker

Code inside the server function will be run once per connection

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



Code outside the server function will be run once per R worker

Code inside the server function will be run once per connection

Code inside of a reactive function will be run once per reaction

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```

Big Data and Shiny

1. **Avoid** unnecessary repetitions

2. **Cache** expensive operations with reactive expressions

A reactive expression is special in two ways

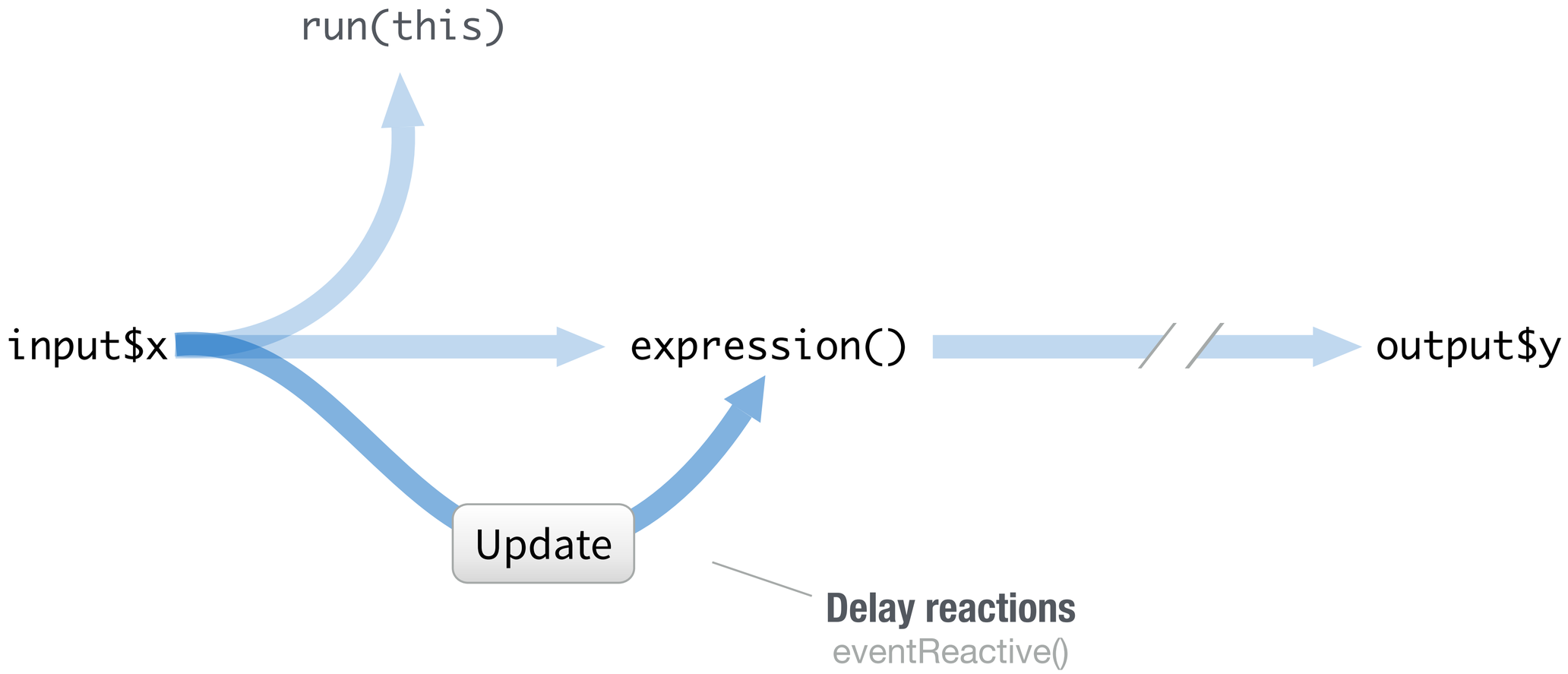
```
data()
```

1. You call a reactive expression like a function

2. Reactive expressions **cache** their values
(the expression will return its most recent value, unless it has become invalidated)

Big Data and Shiny

1. **Avoid** unnecessary repetitions
2. **Cache** expensive operations with reactive expressions
3. **Delay** expensive operations


run(this)
input$x
expression()
output$y
Update
Delay reactions
eventReactive()

eventReactive()

```
data <- eventReactive(input$go, { rnorm(input$num) })
```

Builds an object that:

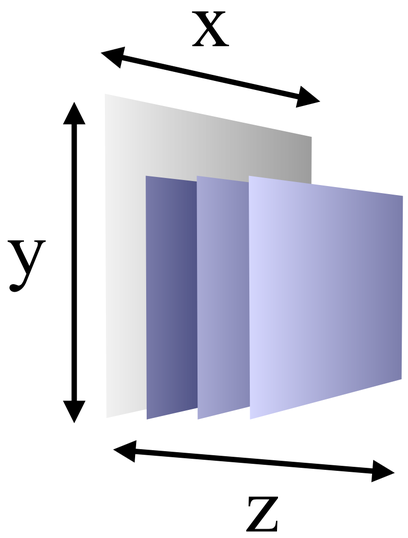
notifies objects that use it that they are invalid

When notified by:

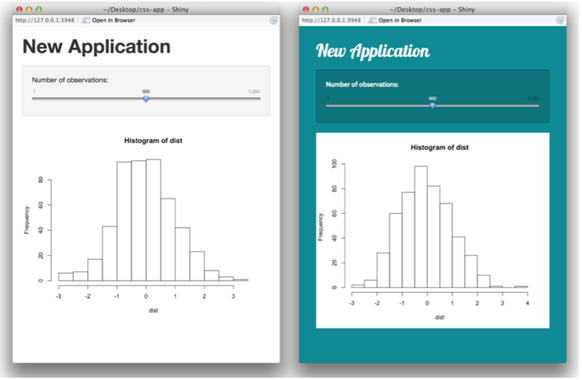
this or these reactive value(s) **and no others**

Where next?

Add static elements



Lay out elements



Style elements with CSS

The Shiny Development Center

shiny.rstudio.com